VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Framework Microsoft ASP.NET MVC 2

Microsoft ASP.NET MVC 2 Framework

2012

Vladimír Vašek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vladimír Vašek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Framework Microsoft ASP.NET MVC 2
Microsoft ASP.NET MVC 2 Framework

Zásady pro vypracování:

Cílem této práce je nastudovat aplikační rámec Microsoft ASP.NET MVC 2, který je nástupcem první implementace návrhového vzoru Model-View-Controller v prostředí ASP.NET. Dále je úkolem studenta popsat postup vývoje webových aplikací na této platformě v prostředí Microsoft Visual Studia 2010, ve kterém je nově nativně podporován, a rovněž popsat výhody oproti klasickému vývoji ASP.NET aplikací. Práce bude dále doplněna o přehled implementací tohoto návrhové vzoru na ostatních platformách a nástrojů, které tento vývoj nativně podporují.
Student následně zužitkuje nabyté znalosti při tvorbě informačního systému pro správu a rezervaci učeben.

Seznam doporučené odborné literatury:

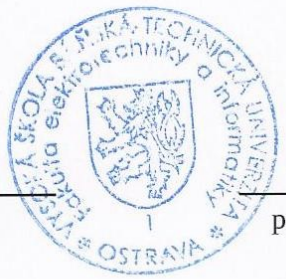
Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Kocyan**

Datum zadání: 19.11.2010

Datum odevzdání: 04.05.2012



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 4.5.2012 Vladimír Vašek

Poděkování

Tímto bych chtěl poděkovat Ing. Tomášovi Kocyanovi za odborné vedení při tvorbě této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá tvorbou webových aplikací v aplikačním rámci ASP.NET MVC 2 od společnosti Microsoft. Cílem práce je podat všeobecný přehled o struktuře aplikačního rámce ASP.NET MVC 2 a jeho vlastnostech. Práce se skládá ze tří hlavních částí. První část podrobněji popisuje návrhový vzor Model-View-Controller, na kterém je postaven celý aplikační rámec ASP.NET MVC 2. Druhá část se věnuje samostatnému aplikačnímu rámci, kde jsou popsány jeho hlavní vlastnosti a jednotlivé moduly odpovídající vzoru MVC. Třetí část se zabývá analýzou a návrhem aplikace pro správu a rezervaci hudeben včetně ukázky implementace.

Abstract

This bachelor thesis discusses creation of web aplications in ASP.NET MVC 2 framework from Microsoft. The goal of this work is to provide an overview of ASP.NET MVC 2 structure and charasteristics. The thesis consists of three main parts. The first part describe design pattern Model-View-Controller in detail, which is built the entire ASP.NET MVC 2 framework. The second part is devoted to ASP.NET MVC 2 framework, which describes its main characteristics and individual moduls corresponding with MVC pattern. Finally the third part deals with analysis and design of aplication for administration and reservation of music hall including demonstration of implementation.

Klíčová slova

Webová aplikace, aplikační rámec ASP.NET MVC 2, návrhový vzor MVC, Model, View, Controller

Key Words

Web aplication, ASP.NET MVC2 framework, MVC design pattern, Model, View, Controller

Seznam použitých symbolů a zkratek

AJAX	- Asynchronous JavaScript and XML
ASP	- Active Server Pages
CLR	- Common Language Runtime
CSS	- Cascading Style Sheets
DLL	- Dynamic Link Library
HTML	- HyperText Markup Language
HTTP	- HyperText Transfer Protocol
IDE	- Integrated Development Environment
JIT	- Just in time
JSF	- JavaServer Faces
JSP	- JavaServer Pages
LINQ	- Language Integrated Query
MSIL	- Microsoft Intermediate Language
MVC	- Model-View-Controller
MVP	- Model-View-Presenter
RIA	- Rich Internet Application
SEO	- Search Engine Optimization
SQL	- Structured Query Language
URL	- Uniform Resource Locator

Obsah

1 Úvod

V dnešní době, kdy připojení na internet je možné téměř odkudkoli, se stále více informačních systémů přesouvá do webového prostředí. Díky tomu se objevují nové aplikační rámce, které mají za cíl co nejvíce usnadnit vývoj webových aplikací a jejich následnou údržbu. Jejich účelem je mimo jiné sjednocení postupu vývoje aplikace tak, aby byl zdrojový kód pochopitelný pro ostatní programátory, kteří jsou s daným aplikačním rámcem seznámeni. Jedním z takových aplikačních rámců je druhá verze ASP.NET MVC od společnosti Microsoft, který má poskytovat všechny výše zmíněné výhody. Tento aplikační rámec bude používán pro následnou implementaci konkrétní webové aplikace pro správu a rezervaci hudebních učeben. Nejdříve je ale nutné se podrobně seznámit s tímto aplikačním rámcem.

Cílem této závěrečné práce je popis druhé verze aplikačního rámce ASP.NET MVC a následná implementace informačního systému pro správu a rezervaci hudeben, který bude postaven na tomto aplikačním rámci.

První část práce se zaměří na popis návrhového vzoru Model-View-Controller, který tvoří základ aplikačního rámce ASP.NET MVC 2. Vzhledem k důležitosti návrhového vzoru MVC bude v dané kapitole podrobněji popsána jeho struktura a variace tohoto návrhového vzoru. Dále budou popsány výhody a nevýhody použití aplikačních rámců využívajících vzoru MVC oproti klasickému vývoji webových aplikací. Tato kapitola také nastíní nejznámější aplikační rámce využívající tento vzor pro tvorbu webových aplikací.

Druhá část závěrečné práce se zaměří na samostatný aplikační rámec ASP.NET MVC 2, kde budou podrobněji popsány jednotlivé části tohoto aplikačního rámce a jeho základní vlastnosti.

Poslední část bude věnována samotné analýze a návrhu aplikace pro správu a rezervaci hudeben. V této části bude zohledněno použití druhé verze aplikačního rámce ASP.NET MVC a bude obsahovat i části kódu naznačující implementaci jednotlivých částí ve vývojovém prostředí Visual Studio 2010 od společnosti Microsoft.

2 Návrhový vzor Model-View-Controller

Návrhový vzor Model-View-Controller (MVC) byl představen v roce 1978 při setkání skupiny vědců věnovaném programovacímu jazyku Smalltalk. První implementaci vzoru vytvořil Trygve Reenskaug, kdy byl vzor MVC implementován jako aplikační rámec pro Smalltalk-80 (knihovna obsahující 80 tříd). Tento aplikační rámec sloužil pro vytváření grafických uživatelských rozhraní. [3]

Avšak s příchodem webových technologií se význam vzoru MVC změnil a tím přizpůsobil práci s webovými aplikacemi. Díky tomu se dnes stal velice rozšířeným architektonickým vzorem a využívá jej mnoho aplikačních rámců, například ASP.NET, Ruby on Rails, Merb, Django a další. Populárním se stal také v technologii Java, kde se používá v aplikačních rámcích Struts, Spring a Tapestry. [2]

Hlavním úkolem návrhového vzoru MVC je oddělení prezentační logiky od doménové logiky aplikace. Výhoda oddělení těchto dvou vrstev spočívá hlavně v lepší udržovatelnosti programu a odděleného vývoje aplikace. To znamená, že zatímco na uživatelském rozhraní může pracovat programátor se znalostmi HTML, CSS a jiných, doménovou logiku může vyvíjet jiný programátor zaměřený na práci s daty a podobně.

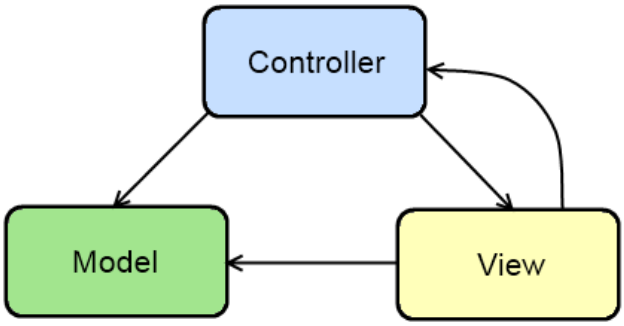
2.1 Základní popis návrhového vzoru Model-View-Controller

Architektura Model-View-Controller se dělí na tři logické části tak, aby je šlo upravovat samostatně a aby byl dopad změn na ostatní části co nejmenší, což umožňuje právě oddělenou správu jednotlivých vrstev, tudíž dobrou udržovatelnost programu. Tyto tři části jsou pojmenovány *Model*, *View* (pohled), *Controller* (řadič). Kvůli přesnějšímu pojmenování jsou nadále užívány originální názvy jednotlivých modulů.

Model je datová struktura (například pole hodnot nebo celá databáze aj.). Obsahuje všechnu aplikační logiku, která není obsažena ve *View* ani v *Controlleru*. Jedná se o validační logiku, doménovou logiku a logiku pro přístup k datům. V některých případech může teoreticky obsahovat pouhou sadu datových objektů bez doménové logiky, ale to je poměrně netypické. [2]

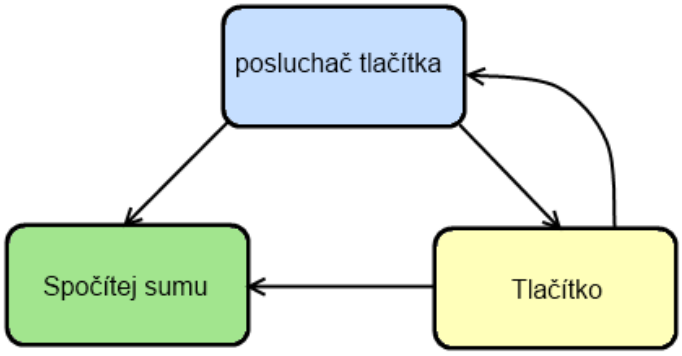
Úkolem části **View** je prezentace dat. *View* je tedy zobrazením *Modelu* a dalších prvků uživatelského rozhraní. Některé prvky uživatelského rozhraní jsou čistě ve *View* a nemusí mít s *Modelem* nic společného – například popisek. [2]

Controller se projevuje až při změně dat v *Modelu* nebo při interakci s uživatelem. V takovém případě má v tu chvíli na starost, aby se aktualizoval *Model* a upravená data přenesl do *View*. *Controller* je ústřední výkonnou jednotkou, která se stará o celkové provázání funkčnosti aplikace. To je obecná definice, která dává velký prostor pro jeho upřesnění. Kvůli tomu mohou vznikat různé variace MVC architektury. [2][3]



Obrázek 1 - schéma vzoru Model-View-Controller

Z obrázku 1 jsou zřejmé jednotlivé vazby mezi jednotlivými moduly vzoru Model-View-Controller. Je zde zobrazena přímá vazba mezi *Controllerem* a *Modelem*, aby mohl *Controller* upravovat data v *Modelu*. Další přímá vazba se vyskytuje mezi *View* a *Modelem*, což umožňuje *View* zobrazit data z *Modelu*. Při některých variacích MVC se často vyskytuje vazba mezi *Controllerem* a *View*, která je v některých případech jednosměrná nebo obousměrná (tyto variace jsou popsány níže v textu). Nikdy však nesmí existovat přímá vazba *Modelu* na ostatní dvě komponenty. V některých schématech se ale může objevit „nepřímá vazba“ z *Modelu* na *View*. Tato nepřímá vazba může reprezentovat notifikační mechanismus (například vzor Observer), který upozorní *View* na změněná data v *Modelu*. Nikdy však *Model* nemůže držet přímý odkaz na *View* nebo na *Controller*. To by bylo hrubé porušení nezávislosti jednotlivých vrstev a zapříčinilo by to špatně oddělenou prezentační a doménovou logiku. [2][3]



Obrázek 2 - reálný příklad vzoru Model-View-Controller

Reálné použití vzoru MVC zobrazuje obrázek 2. Ve vrstvě *View* je generováno tlačítko skrze které lze spustit příslušnou akci. Po kliknutí na tlačítko spustí posluchač tlačítka metodu, která je vyvolána při příchodu události kliknutí na tlačítko (*Controller*) a uvnitř této metody zavolá metodu „spočítej sumu“ z *Modelu*. Tím se změní data v *Modelu*, které mohou být zobrazena zpátky do *View*.

Při detailnějším pohledu na MVC architekturu se mohou vyskytnout komplikace. Například jestli patří prezentační logika do *View* nebo *Controlleru*, kde má být obsažena funkčnost aplikace (*Model* nebo *Controller*), jak úzká vazba je mezi *View* a *Controllerem* nebo zdali má mít *View* přímou vazbu na *Model*. Z těchto důvodu je zde mnoho variací architektury Model-View-Controller, které se snaží jednotlivé komplikace vyřešit. [2]

U návrhového vzoru Model-View-Controller existují dva způsoby interakce uživatele [2]:

- U „widgetových“ systémů (komponentové aplikační rámce typu Java Swing, Windows Forms, Windows Presentation Foundation a podobně) umí uživatelský vstup ošetřit komponenty samy – například tlačítko reaguje na událost kliknutí, textové pole dokáže zachytávat, co uživatel píše na klávesnici a podobně. V takových systémech zachycuje *View* vstup.

- Pak existují systémy „newidgetového“ typu, kde žádné komponenty neexistují a ošetření uživatelského vstupu je tak potřeba učinit někde jinde. V takovém případě zpracovává uživatelský vstup *Controller*.

2.2 Vývoj vzoru Model-View-Controller a Model-View-Presenter

Jak je zmíněno výše, první návrhový vzor Model-View-Controller pochází ze sedmdesátých let dvacátého století. Tehdy na sobě byli vstup a výstup téměř nezávislé (jedno zařízení se staralo o vykreslování na obrazovku a druhé obstarávalo vstupy z klávesnice). Díky tomu vznikl vzor Model-View-Controller, který odrážel stav hardwaru. *View* se zde staralo o vykreslování uživatelského rozhraní a *Controller* obstarával vstupy z klávesnice. [2][3]

S rozvojem operačních systémů a programovacích jazyků postupně zanikal rozdíl mezi vstupem a výstupem. Například tlačítko nyní zvládá jak vykreslení na obrazovku, tak ošetřuje události myši, klávesnice apod. Z tohoto důvodu téměř zanikla potřeba MVC, protože *Controller* neošetřoval vstup. To je také důvod vzniku jeho variace Model-View-Presenter, který není architektonicky moc odlišný, ale principielně ano. Vzor Model-View-Presenter bude popsán níže v textu (kapitola 2.2.2). [2]

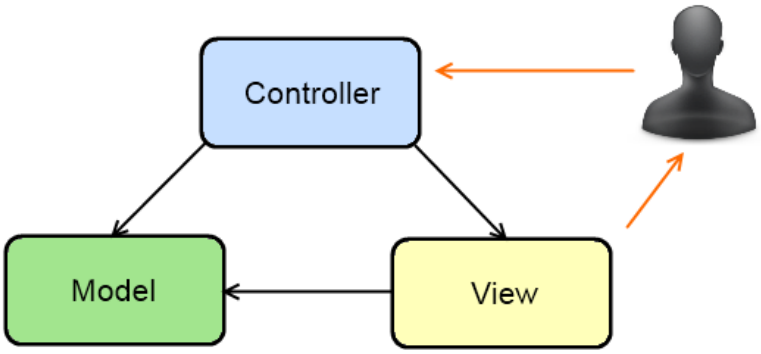
Při nástupu webu v devadesátých letech se zájem o Model-View-Controller prudce zvýšil, protože prostředí webu neumí pořádně sloučit vstup a výstup. U webu je vstupem URL a výstupem se stává stránka s kódem HTML. Proto jsou dnešní aplikační rámce, využívající vzoru Model-View-Controller, v zásadě stejné jako ty původní ze sedmdesátých let. Dnes je jedinou široce rozšířenou platformou s odděleným vstupem a výstupem web, proto se návrhový vzor Model-View-Controller uplatňuje hlavně zde, zatímco klientské technologie jsou téměř všechny komponentové, takže využívají model návrhový vzor Model-View-Presenter. [2]

Na přelomu tisíciletí se objevila webová technologie ASP.NET, která i přes HTTP požadavky umí reagovat na události podobně jako aplikace tlustého klienta (vstup je ošetřen přímo ve *View*). Proto zde MVC opět ztrácí smysl, ale je zde široké využití právě pro model MVP. [2]

Architekturu Model-View-Controller lze použít i u některých aplikací komponentového typu. Tam však nemá *Controller* na starost zpracování uživatelského vstupu, ale typicky se zabývá instrukcemi vyšší úrovně. Může například zachytávat události, které mapuje na nějakou funkcionalitu. *Controlleru* se v tomto pojetí někdy říká aplikační *Controller* a jeho přítomnost se nevylučuje s přítomností *Presenterů*. [2]

2.2.1 Návrhový vzor Model-View-Controller

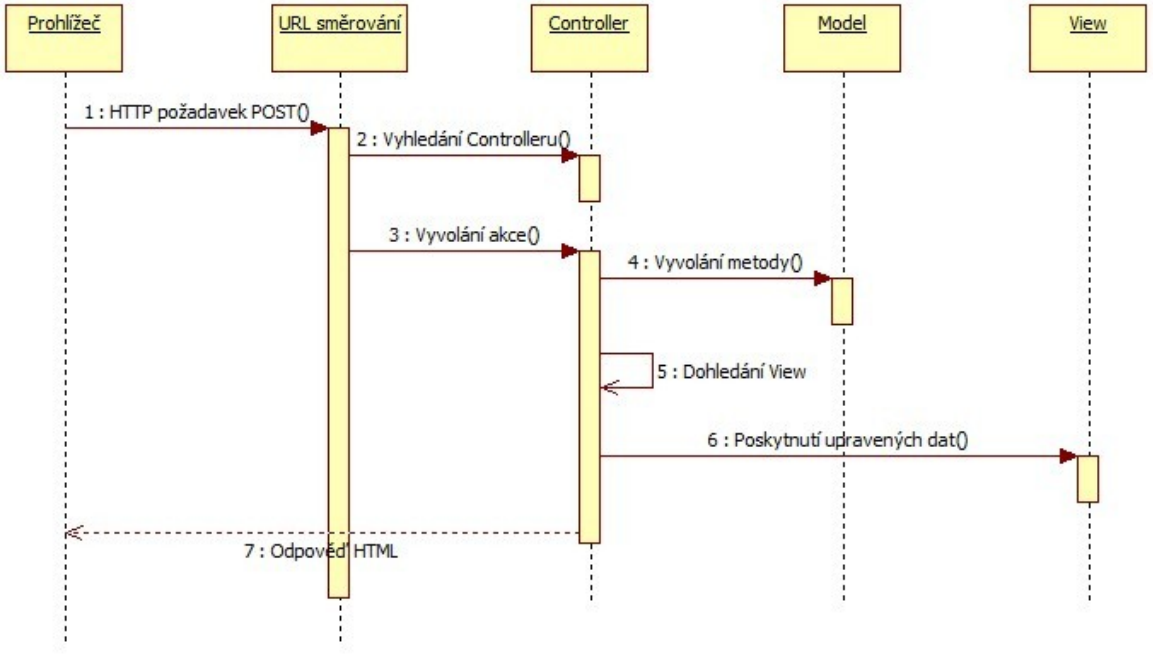
Obrázek 3 popisuje schéma vzoru Model-View-Controller a přístup uživatele k jednotlivým vrstvám vzoru. Jednotlivé šipky zobrazují vztah jednotlivých komponent a uživatele.



Obrázek 3 - Vzor Model-View-Controller a interakce s uživatelem

Tok událostí ve vzoru Model-View-Controller v prostředí webu po poslání HTTP požadavku, který zobrazuje obrázek 4:

1. Po příchodu HTTP požadavku (například typu POST) URL směrovač vyhledá příslušný *Controller* a vyvolá příslušnou akci (*ActionEvent*).
2. *Controller* následně rozhodne jak na tuto akci zareagovat a změní některé hodnoty v *Modelu* nebo přímo ovlivní *View*.
3. *Controller* dohledá odpovídající *View* .
4. *Controller* poskytne *View* patřičná data pro zobrazení (data upravená a získaná z *Modelu*).



Obrázek 4 - sekvenční diagram průběhu vzoru Model-View-Controller

Tento koloběh se stále opakuje pro každý nový HTTP požadavek. Ve schématu se objevuje přímá vazba *Controlleru* na *View*, která je pro MVC typická, ale mnohem důležitější je schopnost *Controlleru* přímo upravit *Model*. Vztah *Controlleru* a *View* je většinou takový, že *Controller* rozhoduje, které *View* se zobrazí uživateli. [2]

Vrstvy Model, View a Controller v prostředí webu

Klasická webová aplikace používá prohlížeč pouze jako jednoduché zobrazovací zařízení, kde se veškeré uživatelské rozhraní generuje na serveru. Naproti tomu plně AJAXová aplikace spoléhá na přítomnost Javascriptu a proto se podstatná část prezentační logiky přesouvá na klienta. V praxi se častěji využívá serverového využití vzoru Model-View-Controller. Je to z důvodů, že plně AJAXových aplikací je zatím málo a částečně AJAXové aplikace nemají tak složitou logiku, aby se vzor Model-View-Controller uplatnil.

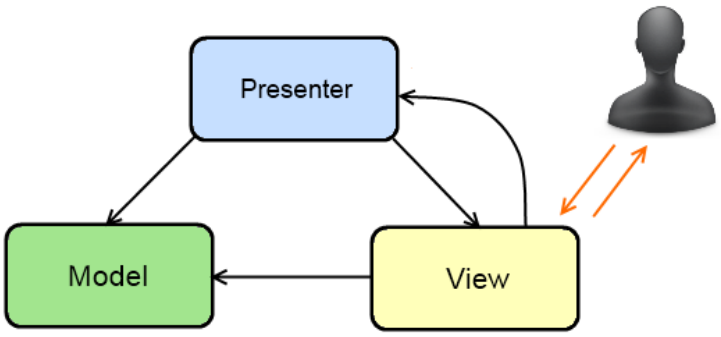
Model je nejjednodušší částí, protože je identický s *Modelem* v aplikacích tlustého klienta, který je popsán výše.

View je serverový kód, který generuje na straně serveru HTML (PHP, Java, C#, Ruby apod.). Ale nemusí se vždy jednat jen o výstup HTML, ale mohou jím být i formáty jako XML a nebo JSON.

Controller je v prostředí webu asi nejsložitější částí. Nejčastěji je rozdělen na dva oddíly. Přední *Controller* (první oddíl) zachytává všechny HTTP požadavky, ty zpracuje a přepošle konkrétním *Controllerům* (druhý oddíl). Konkrétní *Controller* potom přijme data z HTTP požadavku, uloží je do *Modelu* a potom *Model* prováže s *View*, které se už umí postarat o vygenerování HTML. [2][3]

2.2.2 Návrhový vzor Model-View-Presenter

Vzor Model-View-Presenter se využívá nejčastěji u komponentových systémů. Ty se však vyskytují i v prostředí webu. Jak ASP.NET Web Forms, tak řada nových RIA technologií (Flex, Silverlight, JavaFX) jsou jako stvořené pro vzor Model-View-Presenter. Schéma vzoru Model-View-Presenter a jeho interakce s uživatelem je zobrazeno na obrázku 5.



Obrázek 5 - vzor Model-View-Presenter a interakce s uživatelem

Změny vzoru Model-View-Presenter oproti Model-View-Controller:

- Uživatelský vstup i výstup nyní celý kontroluje *View* (tlačítko, textové pole apod.)
- Nejedná se tady o oddělení vstupu a výstupu, ale použití tohoto vzoru je čistě architektonické (pro lepší údržbu)

- *View* má přímou vazbu na *Presenter*, což je důvod zániku oddělení vstupu a výstupu.
- *Presenter* v mnoha případech pracuje s *View*, takže je tato vazba mnohem silnější u běžného vzoru Model-View-Cotroller

Zatímco *Model* je totožný s *Modelem* v architektuře MVC, tak *View* navíc zpracovává uživatelský vstup. V reakci na kliknutí myši zavolá příslušnou metodu na *Presenteru,* který pouze deleguje uživatelské akce. Úloha *Presenteru* se taky mění, protože obsahuje aplikační a prezentační logiku. Pracuje s *Modelem* a pomocí notifikací aktualizuje *View*, nebo přímo ovlivní *View*. [2]

2.3 Výhody Model-View-Controller aplikačních rámců

Obecně lze říci, že každý aplikační rámec nabízí určitou předprogramovanou funkčnost, kterou lze při vývoji webové aplikace využít. Funkce v aplikačním rámci jsou většinou optimalizované a spolehlivé a jejich funkčnost je ověřena v jiných aplikacích. Je proto již dobře otestován a z toho důvodu je i mnohem spolehlivější než jakýkoli vlastní kód. Použitím aplikačního rámce lze tedy snadno zvýšit spolehlivost aplikace. Další výhodou je, že aplikační rámec je v podstatě částečně hotová aplikace a při jeho použití získáme funkční část, kterou si už upravíme a rozšíříme podle našich potřeb.

Oddělení prezentační a datové vrstvy u MVC aplikačních rámců umožňuje snadnější údržbu prezenční vrstvy, ta je díky použití vzoru MVC mnohem jednodušší, tím i přehlednější. Tato výhoda se projeví nejvíce, když je potřeba upravit prezenční vrstvu, což je část, která se ve většině aplikací upravuje nejčastěji. Výhodou striktního oddělení vrstev je také umožnění paralelního vývoje aplikace.

Oddělení jednotlivých vrstev také umožňuje vyměnit implementaci jedné vrstvy za jinou (vždy ale musí jít o stejný typ vrstvy), a to bez vlivu na ostatní vrstvy. Je zde i možnost využívat více prezentačních vrstev na různých platformách, které pracují s jedním *Modelem*, takže každé *View* může data získaná z *Modelu* prezentovat uživateli jiným způsobem, třeba v závislosti na způsobilosti uživatele, nebo na kterém zařízení uživatel pracuje (chytrý telefon, tablet a podobně)

V aplikaci lze nadefinovat jasné a ucelené rozhraní pro práci s *Modelem*. Díky tomu ho mohou ostatní vrstvy snadno používat a manipulovat sním bez znalosti jeho vnitřní struktury. Toto usnadní práci především vývojářům prezenční vrstvy. Použití MVC architektury zjednoduší i budoucí rozšiřování aplikace. Vrstvu *View* a *Controller* lze totiž postupně zvětšovat podle toho, jak se zvětšuje vrstva *Modelu*. [6]

Další výhodou aplikačních rámců využívající vzoru Model-View-Controller je lepší testování aplikace, než je tomu třeba u technologie WebForms. V aplikačních rámcích využívajících vzoru MVC je díky oddělení modulů aplikace lépe testovatelná funkčnost aplikace. U MVC aplikačních rámců se nejčastěji testuje vrstva *Controller*, která obsahuje právě aplikační logiku.

Neméně důležitou výhodou MVC aplikačních rámců je často jednodušší směrování HTML požadavků. O ty se stará *Front-Controller* (přední *Controller*). *Front-Controller* zpracovává požadavky od aplikace pomocí jednoho dalšího *Controlleru.* Ten se dá využít pro vytvoření směrovací infrastruktury. To zapříčiňuje mnohem jednodušší životní cyklus jedné stránky oproti WebForms a jednodušší rozšiřitelnost směrování. [11]

Ve zkratce lze říci, že použitím MVC aplikačního rámce umožňuje rozdělit aplikaci na tři části. Toto rozdělení zjednodušuje vývoj aplikací a tím i snížení chybovosti aplikace. Celkově se tedy zvýší spolehlivost aplikace a zjednodušený vývoj snižuje náklady a zvyšuje rychlost tvorby aplikace.

Zároveň jsou znovupoužitelné jednotlivé části, což způsobuje jednodušší rozšiřování upravované aplikace. [6]

2.4 Nevýhody Model-View-Controller aplikačních rámců

I přes nesporné výhody lze najít i nevýhody spjaté s použitím MVC aplikačního rámce [6]:

Před použitím aplikačního rámce je nutné se jej nejprve podrobně naučit. Jak moc je těžké aplikační rámec používat záleží na konkrétním aplikačním rámci. Většinou platí, že aplikační rámce, které mohou velmi usnadnit vývoj webové aplikace, jsou složitější na pochopení, zatímco jiné aplikační rámce mohou být snadno použitelné, ale při vývoji aplikace nemusí představovat tak velké usnadnění.

Pro použití aplikačního rámce v aplikaci musíme přidat k aplikaci knihovny daného aplikačního rámce. Na vývoji aplikace mohou pak pracovat pouze osoby seznámené s těmito knihovny. Jelikož je MVC aplikačních rámců poměrně velké množství a každý z nich má své výhody i nevýhody, je nutností se dané aplikační rámce často neustále doučovat.

Konkrétní aplikační rámec je skupina tříd, abstraktních tříd a rozhraní, které potom musíme v aplikaci rozšiřovat a implementovat, čímž vzniká poměrně silná vazba aplikace na konkrétní aplikační rámec. Novější aplikační rámce se ale většinou snaží tuto závislost cíleně minimalizovat. Aplikační rámec také často vytváří mnoho pomocných objektů, což může snižovat výkon aplikace.

Architektura MVC nijak neřeší otázku jak a kde jsou data aplikace ukládána. Některé MVC aplikační rámce sice poskytují řešení i těchto otázek, ale to je již nad rámec architektury MVC. [6]

2.5 Přehled aplikačních rámců využívající architektury Model-View-Controller

Následující přehled stručně popisuje nejznámější aplikační rámce, které využívají architektonického vzoru Model-View-Controller. Jedná se o aplikační rámce postavené na různých platformách a psané v různých programovacích jazycích. Jelikož využívají stejné architektury MVC, mají mnoho společného, liší se pouze v zaměření na určité vlastnosti.

2.5.1 Ruby on Rails

Ruby on Rails je jedním z nejpopulárnějších aplikačních rámců současnosti využívající architekturu MVC. Pracuje na základě dvou hlavních zásad. Prvním ze základních principů je „Konvence má přednost před konfigurací“, tedy že programátor konfiguruje pouze ty části aplikace, které se liší od běžného nastavení. Druhým principem je „nejjednodušší řešení je zpravidla nejsprávnější“. Jednou ze silných vlastností Ruby on Rails je jeho použitý jazyk Ruby. Ruby je dynamický programovací jazyk, který podporuje dynamické a úsporné psaní kódu, který je kompilován za běhu překladačem. Ruby, tudíž i Rails, vyvinulo vlastní estetiku pro psaní kódu (vlastní syntaxe). [13][14]

Rails také obsahuje směrování, mapování URL adresy na *Controllery* skrze soubor *config/routes.rb* a poskytuje přístup k datům v databázi pomocí objektově-relačního mapování. Pro objektově-relační mapování využívá návrhového vzoru ActiveRecord, kde se řádky v databázi převedou na instance objektů a sloupce databáze se převedou na jejich atributy. [1]

2.5.2 Django and Python

Django je aplikační rámec využívající jazyk Python a návrhového vzoru Model-View-Controller, který je určený pro rychlý a čistý vývoj webových aplikací. Django se někdy sám nazývá termínem „webový framework pro perfekcionisty s termíny“, protože bylo vyvinuto v novinářském prostředí, pro které bylo důležité provádět úkony spojené s webem rychle a jednoduše.

Tento aplikační rámec je zaměřen primárně na automatizaci tvorby zdrojového kódu a principu neopakuj se (*Dont repeat yourself*), který se snaží vyhnout zbytečnému opakování stejných částí v kódu. Pro urychlení vývoje aplikace umí Django podobně jako Rails automaticky generovat kód administračního prostředí podle databáze. [12]

Django poskytuje taktéž objektově-relační mapování, které umožňuje popsat databázi pomocí jazyka Python. V Djangu se tak pracuje jen s objekty a funkcemi Pythonu, ale umožňuje psát i čisté SQL dotazy, které se mohou hodit pro složitější dotazy.

Stejně jako většina MVC aplikačních rámců pro tvorbu webu obsahuje i Django směrování URL adres na funkce, které se provádí v souboru *urls.py*. [1]

2.5.3 Spring, Struts a Java

V prostředí Java existují tři hlavní aplikační rámce založené na architektuře Model-View-Controller. Jedná se o Apache Struts, Spring a JavaServer Faces, který je standardem pro J2EE (Java 2 Enterprise Editon). Na platformě Java existují i další aplikační rámce založené na architektuře Model-View-Controller, ale pro přehled jsou zde uvedeny pouze tyto tři nejpoužívanější.

Aplikační rámec Spring dodržuje objektově-orientovaný návrh, rozhraní nad třídami a možnosti jednoduchého provádění testů. Spring je aplikační rámec pro objekty doménové vrstvy a i když nevyužívá všech aspektů vzoru MVC, lze jej nalézt v téměř každé aplikaci vytvořené na platformě Java (minimálně na doménové části aplikace). Spring obsahuje aplikační rámec MVC, který poskytuje aplikační kontext pro webové aplikace, což umožňuje jeho spolupráci s ostatními webovými aplikačními rámci platformy Java jako je třeba Struts. [1]

Aplikační rámec Struts je často používaný aplikační rámec, který je stále častěji implementován zároveň se Spring. Struts je převážně založen na technologiích JavaServer Pages, servlety, JavaBeans a proto je i velice spjat s webovým kontejnerem. Tento aplikační rámec nutí programátora striktně dodržovat pevně stanovená pravidla. [13][1]

JavaServer Faces je MVC aplikační rámec poskytující sady standardních, znovupoužitelných grafických komponent pro vývoj webových aplikací. Existuje mnoho různých implementací JSF (JavaServer Faces). Tento aplikační rámec poskytuje tagy komponentového typu pro každý dostupný tag standardního HTML. Obsahuje také validaci dat na straně serveru, převod dat a správu pro navigaci na stránce.

Všechny tři aplikační rámce využívají myšlenky *Modelu*, *View* a *Controlleru*. Lehce se liší v jejich zaměření a životních cyklech, ale hlavní záměr je pořád stejný. Snaží se o oddělení jednotlivých částí.

2.5.4 Zend a PHP

Zend je otevřený aplikační rámec pro webové aplikace implementovaný v jazyce PHP5, který se snaží donést trochu formality do PHP aplikací, ale přitom se snaží zachovat určitou „pružnost“ kódu. Aplikační rámec Zend nevyžaduje používat jeho komponenty, jako je tomu u některých ostatních aplikačních rámců. Zend implementuje *front-controller* model, využívající principu konvence má přednost před konfigurací kdekoliv je to možné. Obsahuje také vlastní šablony pro *View* a také podporuje vložení alternativních *View*.[1]

3 Aplikační rámec Microsoft ASP.NET

Aplikační rámec ASP.NET je produktem společnosti Microsoft a je součástí aplikačního rámce .NET. Tento aplikační rámec je určen pro tvorbu webových aplikací a služeb. Je nástupcem technologie ASP (Active Server Pages) a stal se přímým konkurentem technologie Java JSP (Java Server Pages). [8]

Aplikační rámec ASP.NET je založen na CLR (Common Language Runtime), který sdílí veškeré aplikace postavené na aplikačním rámci .NET. Proto lze programy vyvíjet v jakémkoliv jazyce podporujícím CLR. Mezi tyto programovací jazyky patří například VisualBasic.NET, C#, Jscript, Managed C++ a mutace Perlu, Pythonu apod. [8][10]

Aplikace založené na ASP.NET se neinterpretují, ale kompilují se do jednoho či několika DLL (Dynamic-link library). Kód pro webovou stránku je možné napsat v jakémkoli textovém editoru, potom jej zkopírovat do virtuálního adresáře na webovém serveru, a když klient přistoupí k aplikaci, tak se celá aplikace dynamicky zkompiluje. Zkompilovaná kopie aplikace se uloží do cache paměti pro budoucí potřeby požadavků. Při změně některých souborů se aplikace automaticky překompiluje, jakmile ji bude vyžadovat klient. To je hlavní důvod, proč je aplikace v ASP.NET znatelně rychlejší, než ve skriptovacích jazycích, kde se při každém přístupu klienta znova parsují stránky. [8]

V aplikacích postavených na aplikačním rámci ASP.NET probíhá kompilace ve dvou krocích. V prvním kroku je kód programovacího jazyka (VisualBasic.NET, apod.) zkompilován do přechodného jazyka MSIL (Microsoft Intermediate Language), který se zapíše do souboru zvaného *assembly*. Tato kompilace do MSIL probíhá automaticky při prvním vyžádání stránky nebo jako tzv. předběžná kompilace (*precompiling*). [9][10]

Druhá fáze kompilace proběhne těsně před skutečným vykonáním stránky, kdy se kód MSIL zkomprimuje do nativního nízkoúrovňového strojového kódu. Jedná se o takzvanou kompilaci JIT (just-in-time).

Všechny tyto vlastnosti znamenají pro programátory programujícím v aplikačním prostředí ASP.NET jednodušší přechod mezi vývojem klasické desktopové aplikace a aplikace v prostředí webu. Tento aplikační rámec využívá původních nástrojů a technologií pro vývoj desktopových aplikací a rozšiřuje je do oblasti vývoje webu. Stránky ASP.NET jsou sestaveny z objektů, ovládacích prvků, kterým je možno přidávat vlastnosti a zachytávat na nich události. Toto jsou společné rysy s ovládacími prvky Windows. Při tvorbě webových stránek pak lze použít prvky jako popis (Label), tlačítka (Button) a podobně. Tyto prvky se vykreslují samy do formulářů generováním HTML kódu, který je potom součástí výsledné stránky poslané klientovi. [9]

3.1 Aplikační rámec ASP.NET MVC

ASP.NET MVC je poměrně nový aplikační rámec od firmy Microsoft, která se jeho vývojem zabývá od roku 2007. Vývojáři tohoto aplikačního rámce se snaží o alternativní využití ASP.NET, takže nejde o nahrazení staršího aplikačního rámce WebForms. Stejně jako aplikační rámec WebForms využívá MVC běžných vlastností ASP.NET. Hlavním rozdílem od WebForms je jeho softwarová architektura Model-View-Controller (MVC), která je podrobněji popsána v první kapitole této práce. [1]

Webové servery zpočátku obsahovaly statické HTML uložené ve složce na disku serveru. Postupně rostla důležitost dynamického webu, kde je HTML generováno přímo za běhu z dynamických skriptů, které jsou taky uloženy na disku. U ASP.NET MVC je tato technologie odlišná. Namísto přímé vazby

mezi URL a souboru ležícím na serverovém disku je zde vazba mezi URL a metodou vztahující se na určitý objekt. Směrovací tabulka vybere *Controller* a akci, která se má zavolat. Vybraný *Controller* potom předá uživateli *View* i s daty získanými z *Modelu*. Obsah adresního řádku prohlížeče tak nemusí přesně odpovídat umístění souboru v adresářové struktuře na serveru. To umožňuje použití takzvaného *SEO Friendly URL*, které může v adresním řádku zkrátit a eliminovat ty znaky, které nejsou na české klávesnici, jsou pro uživatele nesrozumitelné, či na něj působí jinak rušivě. [11][1]

ASP.NET MVC má tedy tři komponenty: *Model*, který ověřuje veškerá data aplikace a zajišťuje všechny operace nad nimi, *View* (Pohled), který zobrazuje uživatelské rozhraní, tj. webovou stránku, a *Controller* (Ovládání, nebo také Řadič), který zprostředkovává komunikaci mezi *Modelem* a *View*. V ASP.NET MVC má většinou *Controller* na starosti také validaci dat, které byly získány z *View* do *Modelu*.

Modelem v ASP.NET jsou třídy reprezentující objekty, tyto objekty jsou často data mapovaná z databáze. Dále tyto třídy obsahují kód pro manipulaci s těmito daty. V ASP.NET MVC je to nejčastěji přístupová datová vrstva využívající nástroje jako LINQ to SQL, Entity Framework, NHibernate zkombinovanou s vlastním kódem obsahujícím doménovou logiku. *Model* se v rámci vzoru Model-View-Controller mění asi nejméně, takže ho není třeba dále popisovat. [7][1]

Oproti tomu jsou *View* a *Controller* mnohem složitější částí vzoru Model-View-Controller v ASP.NET. Proto jsou níže podrobněji popsány v samostatných podkapitolách.

3.2 Controller v ASP.NET MVC 2

Controller je nejdůležitější a nejsložitější částí v ASP.NET MVC 2, která řídí celý běh aplikace. Na jejím pochopení spočívá správná implementace celého projektu a dodržení principů vzoru Model-View-Controller.

Podle konvencí aplikačního rámce ASP.NET MVC 2, by měla existovat třída *Controlleru* pro každou podsložku obsahující *View* (nejčastěji aspx stránka). Název třídy takového *Controlleru* se značí jako název této podsložky obsahující *View* s příponou Controller. Například vedle stránky zobrazující seznam zboží na skladě pojmenované jako *SeznamZbozi.aspx* uložené ve složce Zbozi bude existovat třída *Controlleru* označená jako *ZboziController.cs,* která bude obsahovat veřejnou metodu *SeznamZbozi*. Tento *Controller* musí být uložen ve složce Controllers dané aplikace.

Další vlastností všech *Controllerů* v ASP.NET MVC 2 je dědičnost z třídy *System.Web.Mvc.Controller*, která implementuje abstraktní třídu *ControllerBase*. Třída *ControllerBase* dále využívá rozhraní *IController*. Jednotlivé části jsou podrobněji popsány níže od nejjednoduššího a základního rozhraní. [1]

3.2.1 Rozhraní IController

Třída, která se má stát *Controllerem*, musí minimálně implementovat rozhraní *IController*. *IController* je velice jednoduché rozhraní obsahující jednu hlavní metodu *Execute*, která vykoná zadaný požadavek kontextu. Implementací rozhraní *IController* do třídy se docílí možnosti volání třídy skrze adresu URL. Implementace rozhraní *IController* je nejjednodušší způsob, jak vytvořit třídu *Controlleru*, aby ji mohl směrovač správně vybrat. Toto rozhraní je zobrazeno ve zdrojovém kódu 1.[1]

```
Public interface IController
{
        Void Execute(RequestContext regestContext);
}
```

Zdrojový kód 1

Když přijde požadavek skrze HTTP, směrovací systém identifikuje *Controller* podle příslušné URL adresy a ten zavolá metodu *Execute*.

Příklad vytvoření jednoduchého *Controlleru* ve Visual Studiu 2010. Když se vytvoří třída, která implementuje rozhraní *IController*, Visual Studio automaticky vytvoří jednoduchou metodu *Execute*, která vypíše „Hello Word“ do odpovědi. Zdrojový kód 2 zobrazuje automaticky generovaný kód po implementaci rozhraní *IController*.

```
using System.Web.Mvc;
using System.Web.Routing;

public class SimpleController : IController
{
        public void Execute(RequestContext requestContext)
        {
                var response = requestContext.HttpContext.Response;
                response.Write(“<h1>Hello Word!</h1>”);
        }
}
```

Zdrojový kód 2

3.2.2 Abstraktní třída ControllerBase

Kvůli širší využitelnosti třídy *Controlleru* autoři implementovali abstraktní třídu *ControllerBase*, což je základní abstraktní třída pro všechny *Controllery* v ASP.NET MVC 2, která implementuje rozhraní *IController* a je rozšířena o tři hlavní vlastnosti. První vlastností jsou *TempData*, to je slovník pro dočasná data trvající po dobu jednoho požadavku. Druhou vlastností jsou *ViewData*, slovník pro data k zobrazení, a třetí je *ControllerContext* poskytující kontext požadavku specificky upravený pro potřeby MVC. Dále potom obsahuje metody odpovědné za vytvoření tohoto kontextu. [1]

Tato třída je stále velmi jednoduchá a umožňuje vývojářům nízkoúrovňové programování pro konkrétní potřeby *Controllerů*. Co tato třída neumí je schopnost konvertovat akce uživatelů na metody, zde přichází na řadu třída *Controller*, která dědí z této abstraktní třídy.

3.2.3 Třída Controller

Nejčastější cesta pro vytvoření třídy *Controlleru* je zdědění ze základní třídy *System.Web.Mvc.Controller*. Třída *System.Web.Mvc.Controller* je určená jako základní třída pro všechny vytvořené *Controllery* v ASP.NET MVC 2 vytvořené ve vývojovém prostředí Visual Studio 2010. Každá třída *Controlleru* musí být umístěna ve složce Controllers, která je umístěna přímo v adresáři projektu. Každý přidaný *Controller* do této složky automaticky dědí z třídy *System.Web.Mvc.Controller*.

Všechny veřejné metody odvozené z třídy *System.Web.Mvc.Controller*, se stávají metodami akcí, které mohou být volány přes HTTP požadavek. Z HTTP požadavku je určující URL adresa pro zvolení správného *Controlleru* a příslušné metody. V každém *Controlleru* se může takto naimplementovat libovolné množství veřejných metod (metod akcí). Vytvoření třídy Controlleru a jedné metody akce je ukázané ve zdrojovém kódu 3, kde bude výsledná URL spouštějící příslušnou metodu akce vypadat následovně: http://www.domain.com/simple2/goodbye [1]

```
public class Simple2Controller : Controller
{
        public void Goodbye()
        {
                Response.Write("<h1>Goodbye</h1>");
        }
}
```

Zdrojový kód 3

3.2.4 Metoda akce s parametrem

Metody akcí implementovaných v *Controlleru* mohou také obsahovat parametr stejně jako normální metoda desktopové aplikace. Přidání parametru do předchozího příkladu je zobrazené ve zdrojovém kódu 4.

```
public class Simple2Controller : Controller
{
        public void Goodbye(string  name)
        {
                Response.Write("Goodbye>" + HttpUtitlity.HtmlEncode(name));
        }
}
```

Zdrojový kód 4

Tuto metodu spustí následující URL adresa: http://www.domain.com/simple2/goodbye?name=World

3.2.5 Metoda akce s více parametry

Vytvoření metody akce s více parametry je podobné jako vytvoření metody akce s jedním parametrem, při použití řetězce v URL. Ve zdrojovém kódu 5 je uvedena metoda, která vypočítává vzdálenost mezi dvěma body. [1]

```
public void Distance(int x1, int y1, int x2, int y2)
{
        double xSquared = Math.Pow(x2 - x1, 2);
        double ySquared = Math.Pow(y2 - y1, 2);
        Response.Write(Math.Sqrt(xSquared + ySquared));
}
```

Zdrojový kód 5

Při použití výchozího směrování bude požadavek na vyvolání metody akce ze zdrojového kódu 5 vypadat následovně: http://www.domain.com/simple2/distance?x2=1&y2=2&x1=0&y1=0

Druhou možností je definování směrování, které umožní specifikovat parametry v čistějším formátu, který nebude obsahovat nepřehledné znaky (*SEO friendly URL*). To lze udělat uvnitř metody *RegisterRoutes* ve třídě *Global.asax.cs*, kde se pomocí metody *MapRoute* definuje nová cesta. Po přidání nové cesty (zdrojový kód 6) bude URL vypadat následovně: http://www.domain.com/simple2/distance/0,0/1,2 [1]

```
routes.MapRoute(“distance”, “simple2/distance/{x1},{y1}/{x2},{y2}”,
new { Controller = “Simple2”, action = “Distance” });
```

Zdrojový kód 6

3.2.6 Použití volitelného parametru v metodě akce

V první verzi ASP.NET MVC se přidání volitelného parametru provádí vytvořením vlastní cesty s volitelným parametrem, nebo změnou parametru aby akceptoval nulovou hodnotu a dopsáním extra logiky pro tuto situaci. [1]

V druhé verzi ASP.NET MVC se tento problém řeší pomocí *DefaultValueAttribute* z knihovny *System.ComponentModel*. Zde již vyhodnotí samotný aplikační rámec, zdali se jedná o výchozí parametr, takže je možné použít zjednodušeného zápisu volitelného parametru. Tento volitelný parametr se vždy zapisuje jako poslední parametr metody akce (zdrojový kód 7).

```
public void (string name, int count = 10)
{
        string response;
        for (int i = 0; i < count; i++)
        {
                response += (“Hello“ + name + “/n“);
                Response.Write(response);
         }
}
```

Zdrojový kód 7

3.2.7 Návratová hodnota akce (ActionResult)

V předchozích příkladech generují metody akce výsledný text přímo do HTTP odpovědi pomocí metody *Response.Write*. To však není ve většině případů dostačující. Nejčastěji potřebuje metada akce vrátit jiný typ než pouhý text zapsaný do odpovědi. K jednodušší implementaci a větší přehlednosti v kódu slouží právě návratová hodnota akce odvozená z abstraktní třídy *ActionResult*.

V aplikačním rámci ASP.NET MVC je již implementována většina běžně používaných typů odpovědí. Samozřejmostí je možnost definování vlastních typů. Tyto vlastní typy návratových hodnot lze vytvořit jako instance tříd odvozených od abstraktní třídy *ActionResult*. Využití těchto návratových hodnot je doporučené, protože mohou data odpovědi definovat jednoduše a vracet různé typy odpovědí podle situace. [7]

Například pro zobrazení seznamu produktů se vytvoří instance *ViewResult* (derivace *ActionResult*) a naplní se daty z M*odelu* do této instance. Častější je ovšem využití pomocné metody *View* třídy *Controller*. Třída *Controller* obsahuje několik pomocných metod pro navracení instancí návratových hodnot akcí. Tyto metody slouží ke zlepšení čitelnost kódu a snazšímu programování. Metody jsou

často pojmenovány podle typu, který vrací a jsou zkráceny o příponu Result. Například metoda *View* vrací instanci *ViewResult*. Použití metody *View* a její naplnění daty je ukázáno ve zdrojovém kódu 8.[7][1]

```
Public ActionResult ShowProducts()
{
        //teoretický kód
        IList<Product> products = SomeRepository.GetProducts();
        return View(products);
}
```

Zdrojový kód 8

Zde jsou uvedeny typy návratových hodnot akcí, které jsou již obsaženy v aplikačním rámci ASP.NET MVC 2:

- **EmptyResult** – reprezentuje nulovou nebo prázdnou odpověď.
- **ContentResult** – zapisuje specifický obsah jako text přímo do odpovědi.
- **JsonResult** – serializuje objekty do JSON a zapisuje JSON do odpovědi.
- **RedirectResult** – přesměrovává uživatele na danou adresu (URL).
- **RedirectToRouteResult** – přesměrovává uživatele na URL pomocí směrovacího parametru.
- **ViewResult** – vygeneruje do odpovědi odpovídající *View*.
- **PartialViewResult** – podobné jako *ViewResult*, generuje dílčí *View* do odpovědi.
- **FileResult** – základní třída pro následné tři návratové hodnoty akce. Použitelné při vracení souborů uživateli.
- **FilePathResult** – derivuje z *FileResult* a zapisuje soubor do odpovědi na základě cesty souboru.
- **FileContentResult** – derivuje z *FileResult* a zapisuje bitové pole do odpovědi.
- **FileStreamResult** – derivuje z *FileResult* a zapisuje proud do odpovědi.
- **JavaScriptResult** – používá se k vykonání javaskriptového kódu okamžitě na klientovi.

3.3 View v ASP.NET MVC

View je první s čím se koncový uživatel setká. Ani dobře navržený *Model* a ani skvěle vytvořený a přímý *Controller* nezaručí spokojenost koncového uživatele. Uživatel pracuje s aplikací skrze *View*. Takže může být aplikace bohatá na funkce a bez chyb, ale pokud není uživatelsky přívětivé prostředí, zákazník to neocení.

View je zodpovědné za poskytování uživatelských rozhraní uživateli. Je dáno referencí na *Model*, který předělává do formy čitelné pro uživatele. V MVC je veškerá práce nad daty vykonána skrze *Model* a *Controller*, takže jediným úkolem části *View* je předělání výsledků do HTML kódu. Proto by *View* nemělo obsahovat ani aplikační logiku a už vůbec doménovou logiku. *View* by mělo obsahovat co nejméně kódu, ale může v něm být implementována prezentační logika. [3]

3.3.1 Specifikace View

Při dodržení konvencí obsažených v ASP.NET MVC aplikačním rámci je specifikace *View* jednoduchá. Pro každou metodu akce v *Controlleru* vždy existuje soubor (stránka aspx) ve složce View, který má stejný název jako metoda akce. To je základ propojení *View* s metodami akce. Při dodržení těchto konvencí ani není nutné zapisovat název *View*, který má metoda akce v *Controller*

vrátit. *Controller* si automaticky dohledá správné *View* podle názvu a vrátí ho uživateli (zdrojový kód 9).

```
public class HomeController : Controller
{
        public ActionResult Index()
        {
                ViewData[“message“] = “Welcome to ASP.NET MVC!“;
                return View();
        }
}
```

Zdrojový kód 9

Jelikož ve zdrojovém kódu 9 není specifikován název *View* u metody *View*, tak se metoda podívá po *View* se shodným jménem jako je název metody akce. V tomto případě si zvolí /Views/Home/Index.aspx.

Tato konvence může být přepsána a přiřadit metodě akce jiné *View*. Takový případ je uveden v následujícím zdrojovém kódu 10, kde je do původní metody *View* doplněn parametr s názvem vraceného View. *Controller* bude hledat *View* stále ve složce Views/Home, ale zvolí stránku *NotIndex.aspx*. Je-li potřeba použít *View* z jiné složky, použije se tilda syntaxe pro specifikaci celé cesty k *View*. [1]

```
public  ActionResult Index()
{
        ViewData[“Message“] = “Welcome to ASP.NET MVC!“;
        Return View(“NotIndex“);
}
```

Zdrojový kód 10

3.3.2 Silně typové View

Silně typové *View* je typově propojené s *Modelem*. Slouží k lepší čitelnosti kódu, ale hlavně k přesnějšímu propojení *Modelu* a *View*. Jeho použití se doporučuje zvláště díky typové kontrole vkládaných dat, validaci dat a možností plného využití *InteliSense* ve Visual Studiu 2010. [1]

Pro výpis seznamu produktů do *View* se musí udělat následující kroky, aby se vytvořilo silně typové *View*. V metodě akce se specifikuje *Model* pro *View* přetížením metody *View*, do které se zadá *Model* jako parametr (zdrojový kód 11).

```
public ActionResult List()
{
        var products = new List<Product>();
        for (int i = 0; i < 10; i ++)
        {
                products.Add(new Product  {ProductName = “ product “ + i});
        }
        return View(products);
}
```

Zdrojový kód 11

V takovém případě *Controller* hledá silně typové *View* propojené s *Modelem* products. Aby mohl *Controller* dané *View* dohledat, musí se v dalším kroku změnit typ *View*, který standardně dědí z třídy *ViewPage<T>*. To se provede přímo v šabloně daného *View* změnou parametru Inherits jak ukazuje zdrojový kód 12.

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<IEnumerable<Product>>" %>
```

Zdrojový kód 12

Nyní lze přistupovat k vlastnosti *ViewData.Model* s plnou podporou *IntelliSense*. Například:

```
<ul>
<% foreach(Product p in Model) {%>
        <li><%: p.ProductName %></li>
<% } %>
</ul>
```

Zdrojový kód 13

3.3.3 Třídy ViewModels

Často *View* potřebuje zobrazit různá data, které přímo nemapují *Model*. Jedním ze způsobů je využití slovníku *ViewData*. Druhým a lepším způsobem je vytvoření vlastní třídy *ViewModel*. Tato třída může být brána jako *Model*, který existuje pouze pro podporu informací pro *View*. Například pokud existuje stránka nákupního košíku, kde je potřeba zobrazit seznam produktů, celkovou cenu a zprávu uživateli, může se vytvořit třída *ShoppingCartSummaryViewModel,* která je zobrazena ve zdrojovém kódu 14. [1]

```
public class ShoppingCartViewModel
{
        public List<Product> Products { get; set; }
        public decimal CartTotal { get; set; }
        public string Message { get; set; }
}
```

Zdrojový kód 14

Nyní je možné natypovat *View* k tomuto *Modelu* následovně:

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<ShoppingCartSummaryViewModel>" %>
```

Zdrojový kód 15

To nám poskytuje podobné výhody jako silně typového *View* (včetně kontroly typu a *IntelliSense*) bez potřeby nějak zasahovat do tříd *Modelu*. Nejčastěji jsou třídy *ViewModel* používány k zobrazení všech potřebných informací uživateli.

3.3.4 Pomoc s tvorbou HTML

Pro zjednodušení tvorby šablon existuje v aplikačním rámci ASP.NET MVC 2 třída *HtmlHelper*. Ta pomáhá programátorovi usnadnit práci s tvorbou HTML kódu, hlavně její opakující se části. Pro tyto opakující části lze použít metodu třídy *HtmlHelper*, která vypíše kód HTML za programátora. Třída *ViewPage*, ze které standardně každé *View* dědí, obsahuje vlastnost *Html* typu *HtmlHelper*, která dovoluje volat všechny rozšiřující metody vracející řetězec. Zjednodušený zápis odkazu (zdrojový kód 16) s využitím třídy *Html* vygeneruje při kompilaci stránky zdrojový kód 17.

```
<%=Html.ActionLink(“Registruj“,“Register“)%>
```

Zdrojový kód 16

```
<a href=“/Account/Register“> Registruj </a>
```

Zdrojový kód 17

Zápis je téměř stejně dlouhý jako u přímého psaní HTML, výhoda využití třídy *HtmlHelper* spočívá v tom, že je k dispozici automatické doplňování kódu a i když se přesune aplikace na jiné stránky (jiný server), pak bude odkaz stále fungovat.

Standardní nabídka pomocných metod třídy Html v ASP.NET MVC

ActionLink a RouteLink - Obě metody vracejí odkaz na jinou akci v *Controlleru*. *ActionLink* bere jako parametry metodu akce, *Controller* a další parametry akce. *RouteLink* obsahuje jméno cesty a nemá mezi standardními parametry *Controller* ani akci. Ty lze ale vložit pomocí parametru *routeValues*.

BeginForm a EndForm - Vytiskne počáteční a koncový tag formuláře. Parametry určují atributy tohoto formuláře.

TextBox a TextArea - Metody, které se nejčastěji používají ve formuláři, sloužící k vygenerování polí pro zadání textu. Text je určen pro menší záznamy a většinou bývá jednořádkový. *TextArea* je víceřádkové okno pro delší zápisy.

Password - Podobná metoda *TextBoxu*, ale skrývá vepsané znaky do hvězdiček.

Hidden - Vytvoří tag pro ukládání kontextových informací ve *View*, které se nezobrazují v prohlížeči, ale jsou viditelné ve zdrojovém kódu stránky.

Encode - Zkratka metody *HttpUtility.HtmlEncode*, která kóduje daný řetězec do formátu, který může být bezpečně přenášen a zobrazován na internetu.

ValidationMessage a ValidationSummary - Metody používané při validaci dat na straně serveru. Vrací jednotlivé, respektive sumarizující chybové hlášky.

RenderPartial - Metoda vkládající do stránky menší část HTML. Využívá se při násobném použití totožné části kódu nebo ve spojitosti s AJAXem. [7][1]

3.4 Novinky v ASP.NET MVC verzi 2

Aplikační rámec ASP.NET MVC 2 je nativně součástí vývojového prostředí Visual Studio 2010. Nová verze se vyznačuje hlavně větší typovou bezpečností a lepší znuvupoužitelnosti kódu. Přináší s sebou také další výhody při programování oproti první verzi. Pro přehled jsou vypsány níže i s jejich stručným popisem. [1]

RenderAction a Action

Při znovupoužití nějaké části kódu se může použít metody *RenderPartial* ze třídy *Html*. Při použití *RenderPartial* musí rodičovské *View* předávat data částečnému *View*. Pokud je zapotřebí více autonomní částečné třídy *View*, musí si tato třída sehnat data sama. Jelikož je přímý přístup *View* k *Modelu* v rozporu s architekturou MVC, uplatní se zde metoda *RenderAction* nebo *Action*. Ty slouží k vykonání dané metody akce v *Controlleru*. Při specifikaci *Controlleru* a jeho metody akce se aplikační rámec postará o zavolání správné metody, která poskytne *View* potřebná data. *RenderAction* v takovém případě poskytne část kódu, zatímco metoda *Action* ji vrátí ve formě řetězce.

Akční metody v *Controlleru* volané skrze *Html.RenderAction* nebo *Html.Action* se nazývají jako potomci metody akce. Při označení metody akce atributem *ChildActionOnly* se stane tato metoda volatelná pouze metodami *RenderAction* a *Action*. To znamená, že je nebude možné volat jako klasickou metodu akce.

Silně typový HtmlHelper

Nový *HtmlHelper* je silně typový, genericky zná typ *ViewModelu*, což umožňuje použít lepší zápis. Tyto silně typové metody jsou označené příponou For. Například pro zadání hodnoty do *TextBoxu* můžeme nyní použít následující zápis:

`Html.TextBoxFor(m => m.Price)` místo použití řetězce `Html.TextBox("Price")`

Tento zápis přináší mnoho výhod, například kontrola typu proměnné, *IntelliSense* a z toho vyplývající jednodušší předělání aplikace.

ModelMetadata

Třída *ViewDataDictionary* nyní neobsahuje pouze vlastnost *Model*, ale nově také *ModelMetadata* (stejnojmenné třídy), která nese informace o *ViewModelu*. Pomocí *ViewModel.ModelMetadata* se lze dostat k dalším informacím o *ViewModelu*, což umožňuje lepší validaci dat jak na straně serveru, tak na straně klienta.

Templating

Templating vznikl díky vlastnosti *ModelMetada*. *Templating* umožňuje pro každý datový typ (objekt) specifikovat způsob zobrazení pro prohlížení (*DisplayTemplate*) a pro editace (*EditorTemplate*). K tomu slouží metody *Html.DisplayFor* a *Html.EditorFor*.

Asynchronní Controller

Nová třída *asyncController* poskytuje podporu pro asynchronní metody akce. Asynchronní metody akce umožňují dlouhodobou komunikaci třeba s externí webovou službou, bez obsazení vlákna.

4 Vývoj informačního systému pro správu a rezervaci učeben

Tato webová aplikace je určena pro skupinu lidí, studentů ubytovaných na Strahovských kolejích, kteří dobrovolně spravují učebny s hudebními nástroji. Jedná se tedy o neziskovou organizaci, kde jsou všechny členské příspěvky využity pro údržbu, vylepšení vybavení hudebny a další investice. V dnešní době jsou v provozu tři hudebny, v budoucnosti je však možné otevření dalších místností. Každá hudebna má svého správce, který se stará o bezproblémový běh své místnosti, ale může zasahovat i do ostatních.

Jedná se o jednoduchý informační systém umožňující rychlou a přehlednou rezervaci hudeben. Přitom musí být dodrženy určité podmínky pro rezervaci, které budou specifikovány později. Dále bude systém obsahovat jednoduchou správu uživatelů (včetně umožnění či zamítnutí jejich přístupu do hudebny) a hudeben. Součástí systému by měl být i jednoduchý CMS systém na jednoduché přidávání novinek a jiných informací určených pro studenty.

4.1 Funkční specifikace

V této části jsou specifikovány uživatelské role (aktéři), kteří přistupují k danému informačnímu systému, a funkční požadavky na systém, které jsou potom zobrazeny v diagramu případů užití a následně jednotlivě popsány.

4.1.1 Uživatelské role

Uživatele systému je možné rozdělit do dvou hlavních skupin. První skupinu tvoří běžní uživatelé, tedy návštěvníci stránek, kteří nejsou registrovaní. Tento nepřihlášený uživatel má přístup pouze k prezentační části aplikace, to znamená, že si může zobrazit veřejný obsah (prezentace hudeben), ale nemá možnost cokoliv přidávat či měnit v systému. Není mu umožněn ani přístup do rozvrhu hudeben.

Druhou skupinu tvoří přihlášení uživatelé, kteří mají po přihlášení přidělenou určitou roli. V systému existují dvě role a každé jsou přiřazena určitá oprávnění. První rolí je přihlášený běžný uživatel, druhou je administrátor.

- **Přihlášený uživatel**, tedy hudebník, má možnost úpravy svého profilu (správa vlastních kontaktů, změna přihlašovacích údajů), dále má přístup k rezervacím hudebny, které může vytvářet, mazat či upravovat. Tento uživatel má také možnost posílání požadavků administrátorovi, nebo přidání hudebního nástroje, na který hraje.

- **Administrátor** je role, která má nejvyšší pravomoc v rámci celé aplikace. Rozšiřuje běžného uživatele a umožňuje mu zasahovat do všech částí v aplikaci. Role administrátora zahrnuje správu všech uživatelů, správu rezervací (v rozvrhu hodin), možností zamítnutí vstupu uživateli do hudebny (udělení ban), zavření hudebny či změny stavu hudebny. Administrátor také spravuje veškerý obsah na webu, což zahrnuje přidávání novinek (zpráv) či jiných sdělení pro běžného uživatele.

Návštěvník stránek se může registrovat, ale jeho účet musí ověřit a aktivovat administrátor. Poté bude mít stejné pravomoce jako běžný uživatel.

4.1.2 Požadavky na funkcionalitu aplikace

V této sekci jsou popsány všechny požadavky zadavatele na aplikaci vyvíjenou v rámci této závěrečné práce. Pod slovním popsáním požadavků na funkcionalitu aplikace je zobrazen diagram případů užití, který zahrnuje všechny funkce daného systému včetně aktérů, kteří k těmto funkcím mají přístup.

Správa uživatelů

Informační systém obsahuje jednoduchou správu uživatelů. Nový uživatel se zaregistruje, ale nemá přístup ke všem funkcím aplikace. Ty se mu zpřístupní až po ověření registrace administrátorem. Uživatelský účet se stává aktivním až po jeho aktivaci administrátorem. Administrátor tak učiní v případě, že má uživatel zaplacený členský příspěvek a proběhla kontrola zadaných údajů. Při aktivaci administrátor přidělí počet volných hodin, které může uživatel týdně využít k rezervaci hudeben. Za normálních okolností administrátor přidělí čtyři hodiny na týden, avšak tato hodnota může být změněna podle vytížení hudebny.

Administrátor má možnost zasahovat do účtů všech uživatelů, to znamená přidávat, mazat, upravovat uživatele. Pokud uživatel poruší pravidla užívání hudeben, které odsouhlasil při registraci, může mu administrátor zakázat přístup k rozvrhu hodin. Po celou dobu trvání tohoto omezení jsou všechny uživatelovi rezervované hodiny přístupné ostatním uživatelům. Po vypršení omezení může uživatel využívat všech služeb systému.

Aplikace rozvrhu rezervací

Na Strahovských kolejích jsou k dispozici prozatím tři hudebny, kde každá z nich má k dispozici vlastní rozvrh. Hudebny jsou otevřené od pondělí do neděle v čase od devíti do dvaadvaceti hodin. Po jednadvacáté hodině je hudebna přístupná pouze pro akustické nástroje, které nepotřebují zesilovač. Jak už je zmíněno výše, každý uživatel má k dispozici čtyři hodiny týdně. Administrátor může další hodiny přidělit nebo zrušit.

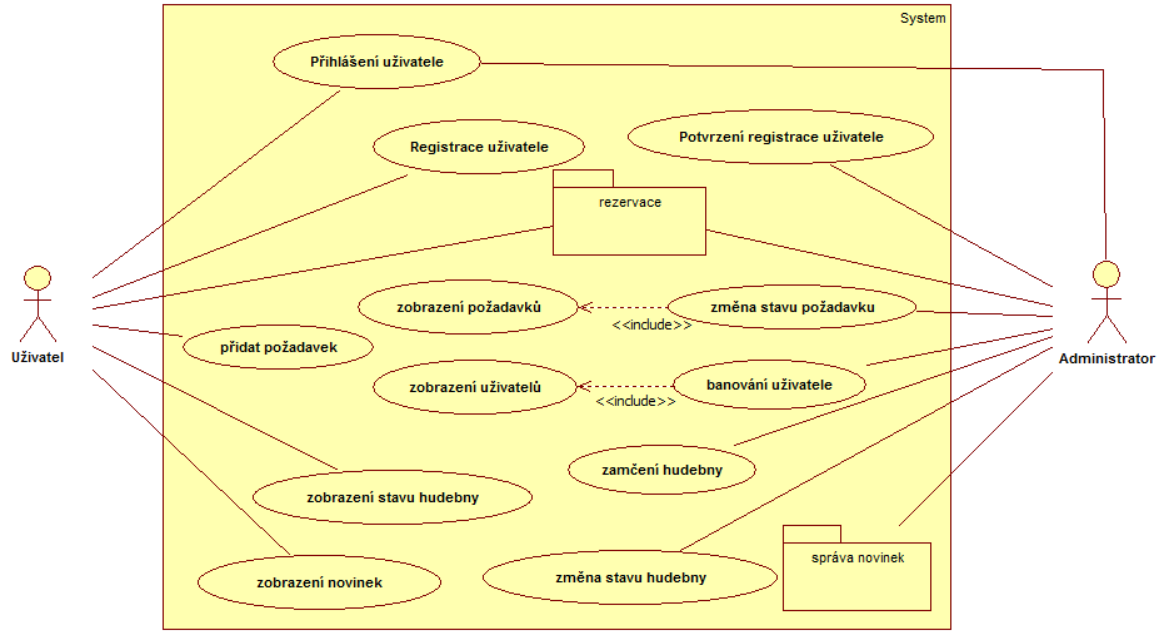
Uživatel, který si zarezervuje hodinu, se stává jejím vlastníkem. Hodina se po zarezervování stává negarantovanou. Tato rezervace se stává garantovanou až po potvrzení rezervace uživatelem. Ten tak musí učinit každý týden minimálně osmačtyřicet hodin před nejbližším začátkem rezervované hodiny. Pokud není rezervace potvrzena uživatelem (vlastníkem), je její potvrzení přístupné ostatním uživatelům v systému.

Rozvrh by měl jít resetovat jedním tlačítkem, čímž se smažou všechny rezervace a obnoví se rozvrh pro nové rezervace. Tuto akci může provést pouze administrátor. Mezi další funkce, které zahrnuje tato část aplikace, patří blokování rozvrhu.

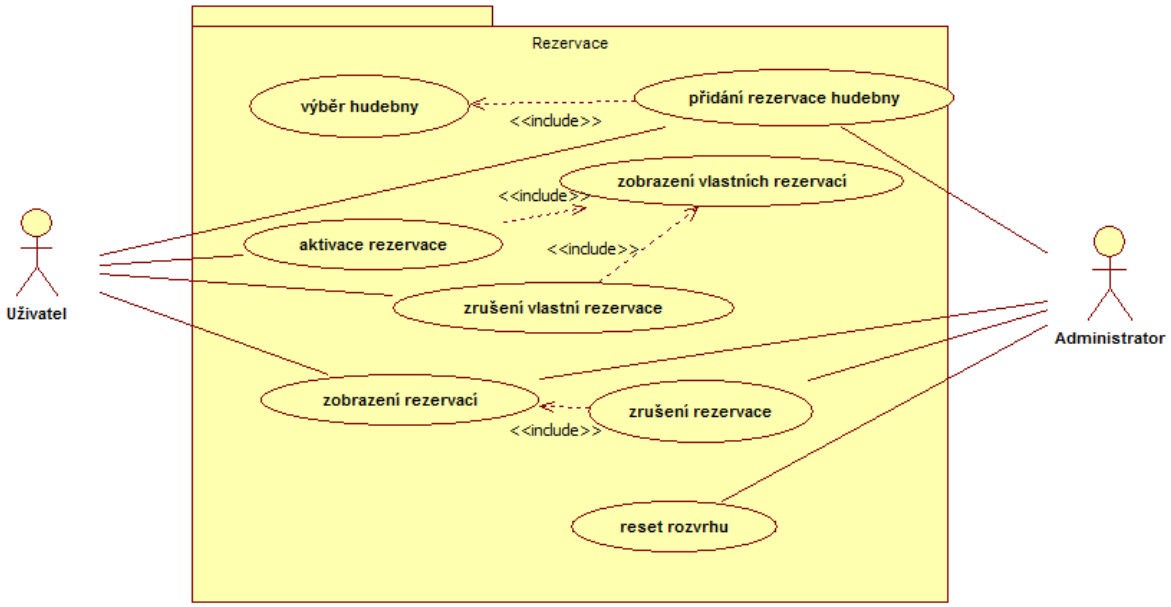
Systém požadavků a správa novinek

Aplikace umožňuje registrovaným uživatelům posílat žádosti administrátorům. Ty se mohou týkat například vadného stavu hudeben, nového vybavení a podobně. K požadavku musí být připojen popisek. Po vytvoření požadavku se sleduje vývoj řešení statusy požadavku (například nový, řeší se, neschválený, vykonaný). Úkolem systému je také informovat uživatele o novinkách, které může přidávat, mazat nebo upravovat administrátor.

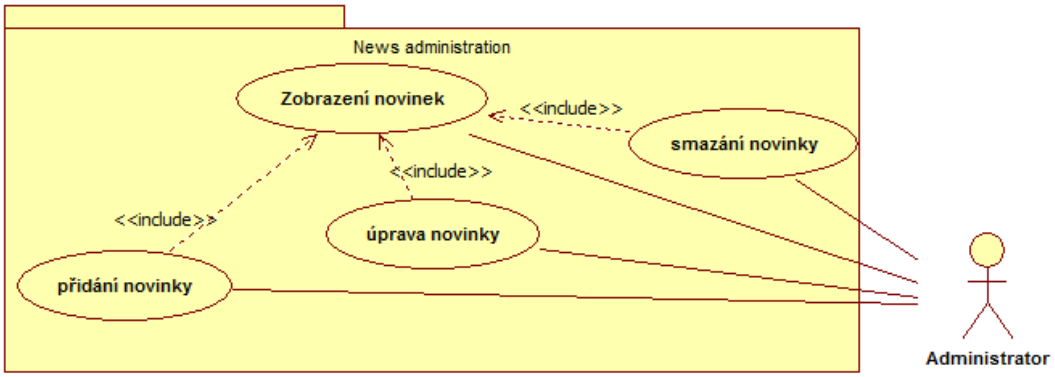
Celá funkcionalita systému včetně pravomocí jednotlivých uživatelů je zakreslena v následujícím diagramu případu užití.



Obrázek 6 - souhrnný diagram případů užití pro webovou aplikaci



Obrázek 7 - diagram užití pro balíček rezervací



Obrázek 8 - diagram případu užití pro správu novinek

4.1.3 Popis jednotlivých případů užití

V této kapitole jsou podrobněji popsány hlavní případy užití pro rezervaci hudeben. Ostatní případy užití jsou umístěny v příloze.

Zobrazení rezervací	
Seznam aktérů:	Uživatel, Administrátor
Prekondice:	1. Musí být zadáno datum, pro určení týdne zobrazení rezervací. 2. Musí být vybrána hudebna, pro kterou má být rozvrh zobrazen
Postkondice:	
Typický průběh:	Systém zobrazí rezervace vybrané hudebny v podobě týdenního rozvrhu.
Alternativní průběh:	Systém zobrazí informace uživateli při omezení hudebny (uzamčení a podobně).
Priorita:	Vysoká
Četnost užití:	Mnohokrát za hodinu

Tabulka 1 - případ užití zobrazení rezervací

Vytvoření rezervace	
Seznam aktérů:	Uživatel, Administrátor
Prekondice:	1. Uživatel musí mít v daném týdnu volné hodiny k rezervaci. 2. Uživateli nesmí být zakázán přístup 3. Uživatel musí mít zaplacený členský příspěvek na daný semestr 4. Datum rezervace musí být ve školním semestru.
Postkondice:	1. V databázi je vytvořena rezervace s neaktivovaným statusem.
Typický průběh:	Uživatel zadá datum a čas, vybere hudebnu a systém ověří možnost vytvoření rezervace (zdali jsou splněny všechny podmínky pro vytvoření rezervace).
Alternativní průběh:	Systém vypíše chyby, proč nemůže být rezervace vytvořena.
Priorita:	Vysoká
Četnost užití:	Denně

Tabulka 2 - případ užití vytvoření rezervace

Aktivace rezervace	
Seznam aktérů:	Uživatel, Administrátor
Prekondice:	1. Uživatel musí mít vytvořenou rezervaci. 2. Aktivace lze provést nejdříve sedm dní před začátkem platnosti rezervace
Postkondice:	V databázi je u rezervace změněn stav z neaktivovaná na aktivovaná.
Typický průběh:	Uživatel potvrdí rezervovanou hodinu, aby se stal jejím vlastníkem.
Alternativní průběh:	Při nepotvrzení rezervace minimálně osmačtyřicet hodin dopředu rezervace propadá a mohou ji aktivovat ostatní uživatelé.
Priorita:	Vysoká
Četnost užití:	Denně

Tabulka 3 případ užití aktivace rezervace

4.2 Technická specifikace

Platforma

Informační systém rezervace hudeben bude realizován na platformě .NET. Na této platformě bude využito druhé verze aplikačního rámce ASP.NET MVC, který je nativně podporován ve vývojovém prostředí Visual Studio 2010. Pro celou aplikaci bude použit programovací jazyk C#, který poskytuje přehledné objektově orientované programování a je jedním z jazyků podporovaným platformou .NET.

Databáze

Pro potřeby aplikace bude využita MSSQL databáze od společnosti Microsoft, která poskytuje všechny potřebné nastavení a umožňuje jednoduchou implementaci do projektu. Navíc je také podporována aplikací Visual Studio 2010, kde lze tabulky jednoduše vytvořit a naplnit počátečními

daty. Systém řízení báze dat MSSQL má pokročilé funkce, které mohou být využity pro pozdější rozšíření aplikace.

Databáze obsahuje čtrnáct tabulek, z čehož jsou čtyři číselníky a jedna vazební tabulka mezi uživatelem a hudebními nástroji, které uživatel využívá.

Počet přistupujících uživatelů

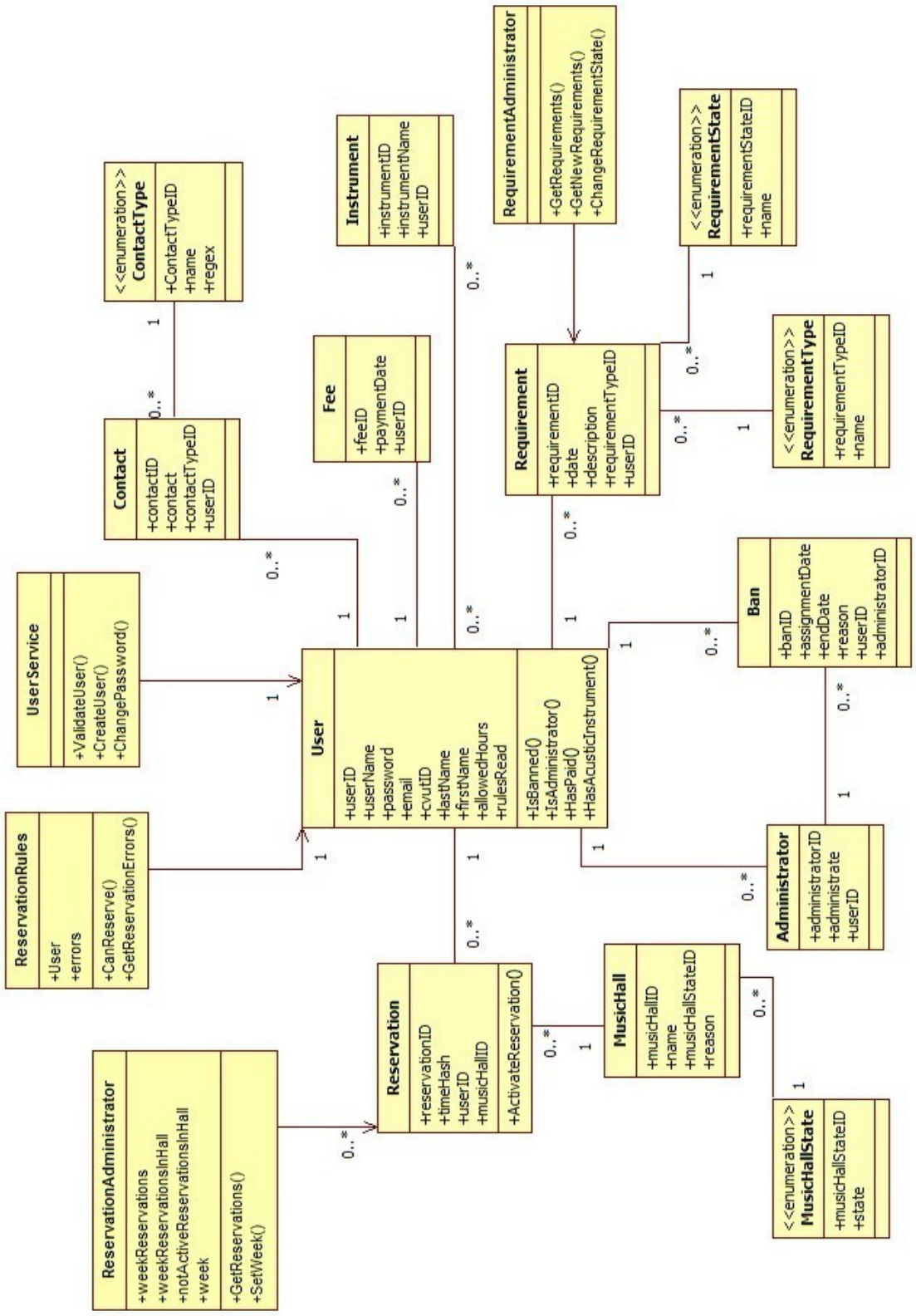
Aplikace je určena pro úzký kolektiv lidí, kteří tyto hudebny využívají nebo spravují. Jelikož se jedná pouze o studenty vysoké školy, kteří jsou ubytovaní na Strahovských kolejích, tak by celkový počet uživatelů neměl přesáhnout hranici osmdesáti uživatelů. O větším počtu uživatelů se v dohledné době ani neuvažuje kvůli omezené kapacitě učeben.

Bezpečnost

Systém bude zabezpečen pomocí autentizace uživatele. Ta bude prováděna přímo na stránkách aplikace pomocí uživatelského (přístupového) jména a hesla. Zadané přihlašovací údaje se ověří pomocí SQL databáze. Po autentizaci uživatele následuje autorizace, která přidělí danému uživateli určitá práva. Dalším zabezpečením systému je ověření registrace uživatele administrátorem.

4.3 Analýza doménového modelu

Po specifikaci požadavků, ať už funkčních nebo technických, je možné nastínit doménový model aplikace. To je hlavní jádro aplikace, které ovlivňuje její chování a má za úkol dodržovat požadavky aplikace. Doménový model zobrazuje třídy, které budou využity při následující implementaci informačního systému. Doménový model je zobrazen pomocí následujícího třídního diagramu (obrázek 9).



Obrázek 9 - třídní diagram doménového modelu

5 Návrh informačního systému pro rezervaci a správu hudeben

Po analýze informačního systému je možné udělat podrobnější návrh aplikace, kde již budou využity jednotlivé vzory, které budou použité v aplikaci pro správu a rezervaci hudeben. Tento návrh také zohledňuje využití aplikačního rámce ASP.NET MVC 2 a dodržuje jeho strukturu.

5.1 Použité vývojové prostředí, aplikační rámce a návrhové vzory

V této části je souhrn použitého vývojového prostředí, aplikačních rámců a vzorů, které byly užity pro vývoj aplikace včetně krátkého popisu a důvodů využití.

Prostředí Visual Studio 2010

Pro implementaci bude použito vývojového prostředí Visual Studio 2010 od společnosti Microsoft. Hlavním důvodem, proč bude použito tohoto vývojového prostředí je, že nativně podporuje druhou verzi aplikačního rámce ASP.NET MVC, na kterém je postavena celá aplikace. Ve Visual Studiu si stačí vybrat programovací jazyk, ve kterém má být projekt vyvíjen, a vytvořit nový webový projekt ASP.NET MVC 2. Následně Visual Studio vytvoří projekt se striktně danou adresářovou strukturou, která dodržuje architektonický vzor Model-View-Controller.

Tento software také napomáhá s dodržením konvencí pro pojmenování tříd. Tato vlastnost je přístupná při využití aplikačního rámce ASP.NET MVC. Například při přidání třídy do složky Controllers doplní automaticky za název třídy příponu Controller. Podobně je tomu i při tvorbě *View*, kdy stačí kliknout pravým tlačítkem do metody akce, pro kterou je potřeba vytvořit *View*, a vybere se možnost vytvořit View. Visual Studio ve složce Views vytvoří podsložku (pokud ještě neexistuje) pojmenovanou shodně s názvem třídy *Controlleru*, do které vloží dané *View* se shodným názvem metody akce. Tím je zajištěno základního propojení mezi moduly a dodržení konvencí pro aplikační rámec ASP.NET MVC 2.

Visual Studio obsahuje také jednoduché IDE pro tvorbu a správu SQL databáze, kde lze jednoduše vytvořit celou databázi včetně všech procedur, funkcí, pohledů a podobně. Toto IDE bude použito hlavně pro vytvoření databáze a naplnění počátečních dat.

Aplikační rámec .NET čtvrté verze

Aplikační rámec .NET je určen primárně pro operační systém Microsoft Windows, kde jeho jednotlivé verze jsou součástí Windows. Čtvrtá verze byla vyvíjena společně s vývojovým prostředím Visual Studio 2010 a je doporučenou součástí operačního systému Windows 7. Obsahuje obsáhlé knihovny, které poskytují otestované a znovupoužitelné kódy pro všechny hlavní oblasti vývoje aplikace.

Aplikační rámec ASP.NET MVC 2

Tento aplikační rámec je podrobněji popsán v třetí kapitole této práce, proto ho zde nebudu dále rozepisovat. V rámci tohoto aplikačního rámce je také použito návrhového vzoru Model-View-Controller, který je podrobněji popsán v první kapitole této práce.

Aplikační rámec Entity Framework

Tento aplikační rámec slouží pro mapování SQL relační databáze. Umožňuje jednoduchý přístup do databáze, převádí data na objekty, se kterými lze libovolně pracovat. Při práci s objekty zůstává

v pozadí aplikace a sleduje změny. Tohoto aplikačního rámce bude použito kvůli možnostem budoucího rozšíření aplikace a kvůli jeho dobrému objektově-relačnímu mapování.

Alternativou pro tento aplikační rámec by mohl být LINQ to SQL. Po přečtení diskuzí pro porovnání těchto dvou aplikačních rámců jsem se ale rozhodl pro Entity Framework, který se pro strukturu mé databáze hodí více. LINQ to SQL se používá pro jednodušší databáze s jednoduchými vazbami mezi tabulkami (jedna ku jedné), zatímco Entity Framework podporuje i složitější vazby. Kromě toho má Entity Framework větší podporu od vývojářů pracujících u společnosti Microsoft.

Návrhový vzor „repozitář“

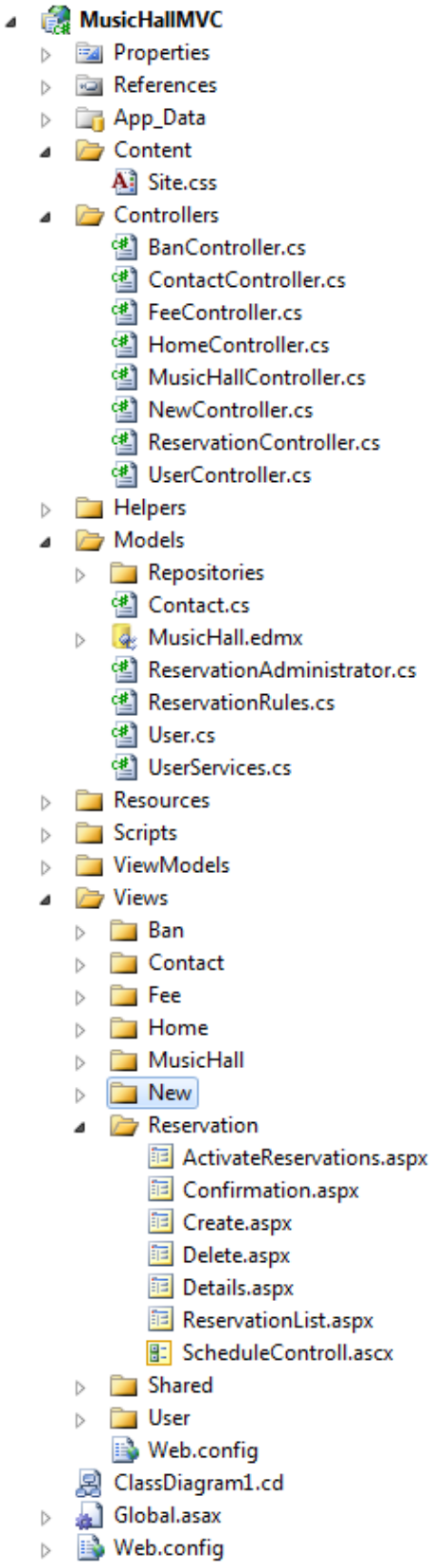
Tento vzor je použit společně s aplikačním rámcem Entity Framework. Jeho úkolem je provádět operace nad daty v databázi. Pro každou tabulku v databázi existuje třída „repozitář“, která umožňuje vyhledávání, vkládání, mazání dat v dané tabulce. Také obsahuje metodu pro uložení změn v databázi. Tímto je oddělena vrstva přístupu k databázi. Tudíž se může kdykoliv nahradit typ databáze bez nutnosti opravovat vrstvu modelu. Pouze se upraví třídy „repozitář“.

5.2 Adresářová struktura aplikace

Adresářová struktura aplikace je vytvořena podle potřeb aplikačního rámce ASP.NET MVC 2. Proto je koncipována tak, aby promítala architekturu Model-View-Controller. Hlavní část aplikace tak tvoří složky pojmenované po jednotlivých modulech architektury Model-View-Controller. Jedná se o složky Models, Views, Controllers, do kterých jsou vkládány soubory podle toho, kterému modulu z dané architektury odpovídají. Tato hierarchie a názvy adresářů musí být dodrženy pro správný chod aplikace.

Kromě těchto tří hlavních složek obsahuje aplikace další volně rozšiřitelné nebo upravitelné složky obsahující používané knihovny a podobně. Následuje přehled hlavních složek a jejich uspořádání adresářová struktura aplikace je zobrazena v obrázku 10.

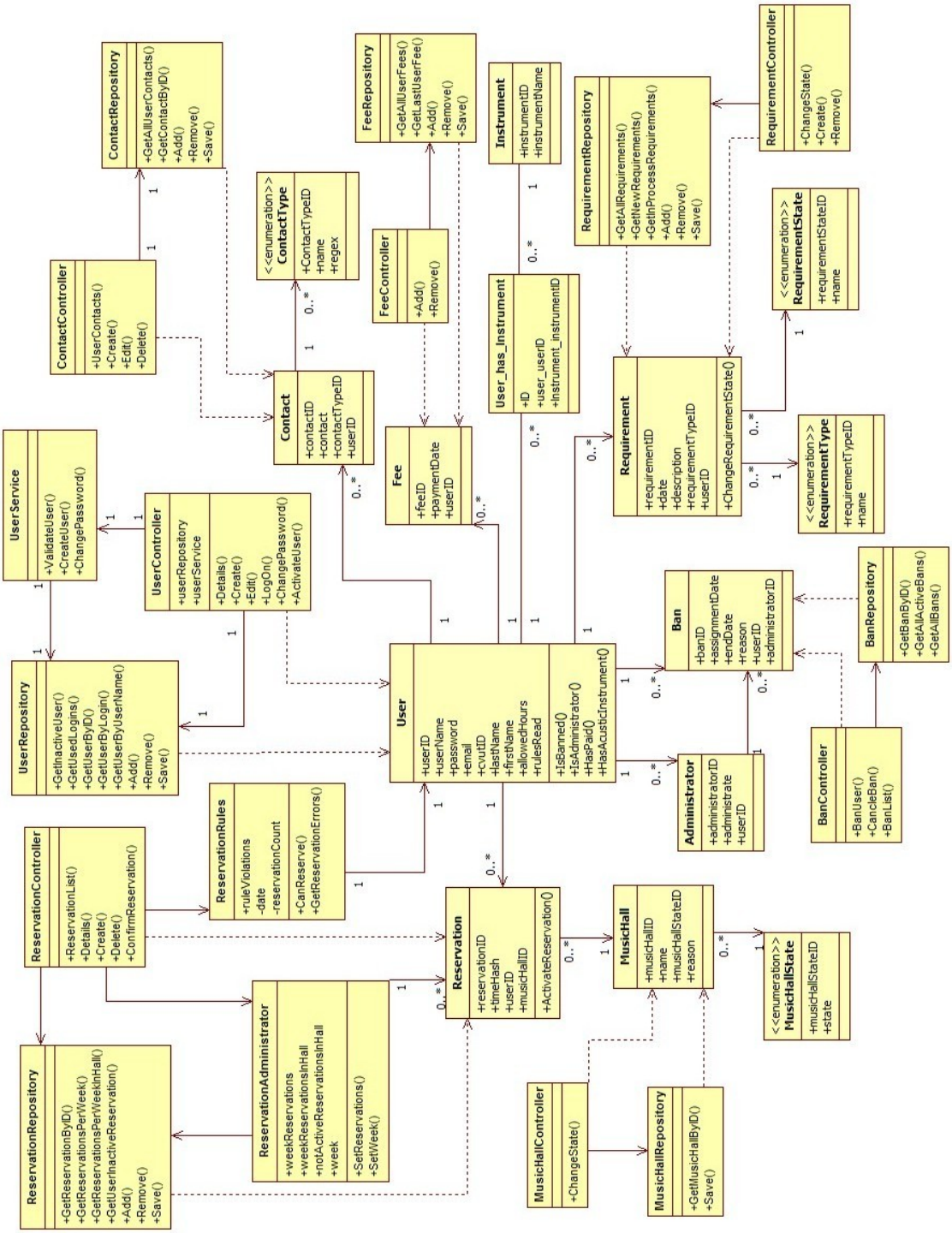
- Content – Složka obsahující soubor s kaskádovými styly.
- Controllers – Umístění všech *Controllerů* aplikace, kde každá třída v této složce obsahuje příponu Controller, aby jej mohl aplikační rámec identifikovat.
- Helpers – Tato složka obsahuje statické třídy obsahující pomocné metody pro *Controllery*. Například metodu pro přidání chybných zpráv do slovníku.
- Models – Složka Models obsahuje třídy reprezentující veškerou doménovou logiku včetně objektově-relačního mapování databáze.
- Models/Repositories – Složka obsahující třídy podle vzoru „repozitář“, které komunikují s databází (získávají, upravují, vytváří nebo mažou data z databáze).
- Scripts – Složka obsahující knihovny pro skriptovací jazyky.
- ViewModels – Složka s třídami uchovávajícími data pouze pro potřeby *View*. Nejsou součástí modelu.
- Views – Složka obsahující všechny dynamicky tvořené stránky. Jsou v ní vytvořeny podsložky, které mají shodný název s *Controllerem*, který tyto pohledy využívá. V těchto podsložkách jsou teprve umístěny samostatné dynamicky generované stránky pro každou metodu akce *Controlleru*. Dále je ve složce *Views* umístěna podsložka Shared, kde jsou uloženy společné části pro všechna *View* (hlavní stránka a podobně)



Obrázek 10 - adresářová struktura aplikace

5.3 Diagram tříd informačního systému

Následující diagram tříd popisuje strukturu aplikace s využitím architektonického vzoru Model-View-Controller a vzoru pro získávání dat z databáze „repozitář“. Proto je ke každé entitní třídě přidělena třída *Controlleru* a repozitáře, které jsou závislé na daném objektu.

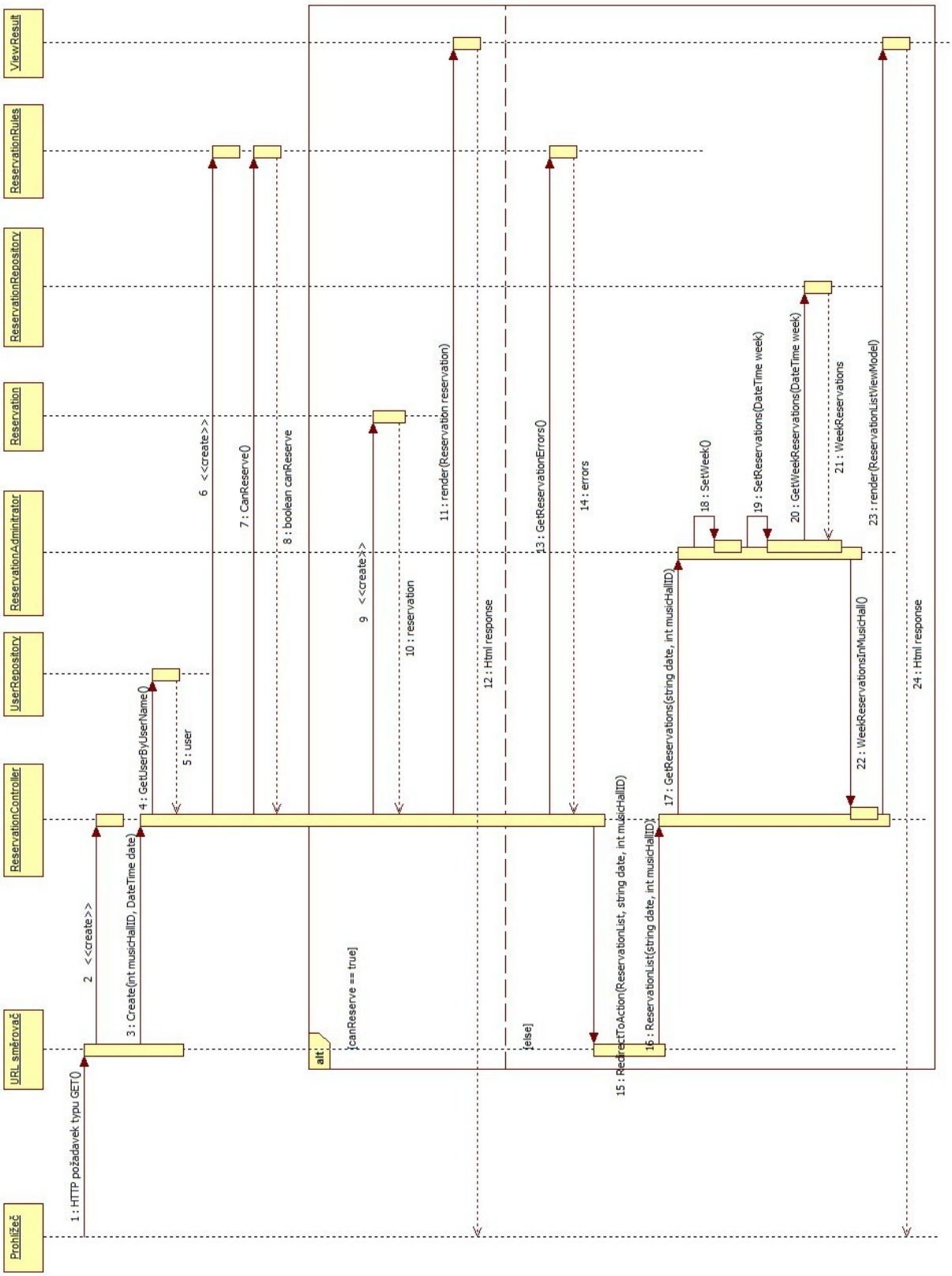


Obrázek 11 - Diagram tříd (návrh)

5.4 Sekvenční diagram průběhu vytvoření rezervace v ASP.NET MVC 2

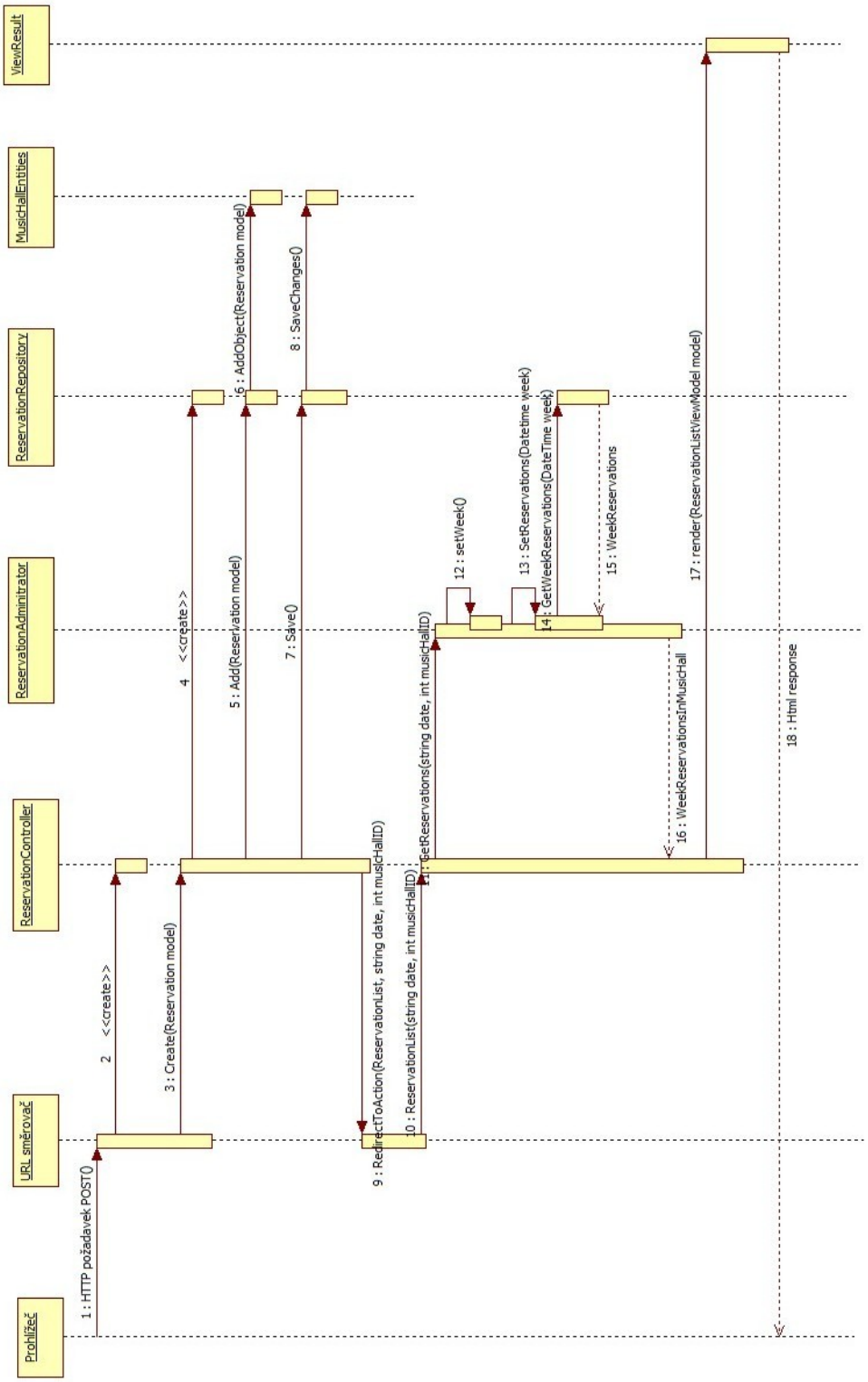
Pro upřesnění průběhu vyřízení HTTP požadavku v aplikačním rámci ASP.NET MVC 2 je zde uveden příklad vytvoření rezervace v informačním systém. Tento sekvenční diagram zobrazuje také metodu akce pro zobrazení rezervací v daném týdnu, která je zavolána po úspěšném uložení rezervace do systému.

První příchozí HTTP požadavek pro vytvoření rezervace je typu GET, který generuje stránku pro potvrzení vytvoření rezervace. Následující sekvenční diagram (obrázek 12) popisuje vyřízení tohoto požadavku.



Obrázek 12 - sekvenční diagram po vytvoření rezervace (GET)

Níže uvedený sekvenční diagram zobrazuje vyřízení HTTP požadavku typu POST, který je volán po potvrzení formuláře pro vytvoření rezervace.



Obrázek 13 - sekvenční diagram pro vytvoření rezervace (POST)

5.5 Implementace třídy ReservationController

Následující zdrojový kód ukazuje část implementace třídy *ReservationController*, která obsahuje metody pro vytvoření rezervace popsané v sekvenčním diagramu (obrázek 12 a 13).

```
public class ReservationController : Controller
{
        //
        // GET: /Reservation/Create

        public ActionResult Create(int musicHallID, DateTime time)
        {
                reservationRules = new ReservationRules(User.Identity.Name, time);

                if (reservationRules.CanReserve(User.Identity.Name))
                {
                        Reservation reservation = new Reservation();
                        reservation.userID = reservationRules.User.userID;
                        reservation.musicHallID = musicHallID;
                        reservation.timeHash = time;

                        return View(reservation);
                }
                else
                {
                        return RedirectToAction("ReservationList",
                                new { date = time.Date.ToString("d"), musicHallID = musicHallID});
                }
        }

        //
        // POST: /Reservation/Create

         [HttpPost]
        public ActionResult Create(Reservation model)
        {
                try
                {
                        reservationRepository = new ReservationRepository();
                        model.activated = false;
                        reservationRepository.Add(model);
                        reservationRepository.Save();
                        User user = userRepository.GetUserByID(model.userID);

                        return RedirectToAction("ReservationList",
                                new { date = model.timeHash.Date.ToString("d"),
                                musicHallID = model.musicHallID});
                }
                catch
                {
                        return View(model);
                }
        }
}
```

Zdrojový kód 18

První metoda akce v daném *Controlleru* se spouští s příchodem HTTP požadavku typu GET. Adresa tohoto požadavku musí obsahovat dva parametry pro správné spuštění metody akce. Prvním parametrem je číslo hudebny, pro kterou má být rezervace vytvořena, druhým parametrem je datum a čas, na který si chce uživatel hudebnu rezervovat. V této metodě akce ověří třída *ReservationRules* (třída tvořící model aplikace), zdali je rezervace možná (jestli jsou splněny všechny podmínky pro vytvoření rezervace) a podle výsledku navrátí *View* s naplněnými daty z *Modelu*. Při splnění podmínek vrátí *View*, který obsahuje formulář pro potvrzení rezervace hudebny. Při porušení pravidel se vrátí na výpis seznamu rezervací hudebny v daném týdnu. Toto *View* by se mělo postarat o vypsání důvodů, proč nelze rezervovat hudebnu na daný čas.

Druhá metoda akce je zavolána při příchodu HTTP požadavku typu POST, tedy po potvrzení formuláře pro vytvoření registrace. Jelikož se jedná o silně typové *View*, které *Controlleru* předalo data, obsahuje tato metoda parametr typu *Reservation* místo typu *FormCollection*. V této metodě proběhne uložení rezervace do databáze. V případě, že uložení do databáze proběhne v pořádku, vrátí metoda odpovídající *View* se seznamem rezervací v daném týdnu. Pokud byly porušeny některé z validačních pravidel, vrátí zpět aktuální stránku i se seznamem chyb, které vznikly při ukládání rezervace do databáze.

5.6 Implementace šablony pro vytvoření rezervace

Následující kód zobrazuje implementaci částečné šablony (*PartialView*), která generuje stránku pro potvrzení vytvoření registrace hudebny.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<MusicHallMVC.Models.Reservation>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Vytvoření rezervace
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <%: Html.ValidationSummary(true) %>

    <% using (Html.BeginForm()) {%>

        <fieldset>
            <legend>Shrnutí rezervace</legend>

            Opravdu chcete zarervovat hudebnu na
            <strong>
                <%: Model.timeHash.ToString("dddd",
                System.Globalization.CultureInfo.CreateSpecificCulture("cs-CZ")) %>
            </strong>
            v čase
            <strong>
                <%: Model.timeHash.ToString("HH:mm") %> -
                <%: Model.timeHash.AddHours(1).ToString("HH:mm")%>
            </strong>?

            <%: Html.HiddenFor(model => model.timeHash) %>
            <%: Html.HiddenFor(model => model.userID) %>

            <br /><br />

            <input type="submit" value="Rezervovat" />
        </fieldset>

    <% } %>

    <div>
        <%: Html.ActionLink("Zpět na seznam rezervací", "ReservationList") %>
    </div>

</asp:Content>
```

Zdrojový kód 19

Jedná se o silně typové *View*, které je vráceno na konci akční metody *Create* z třídy *ReservationController*. *View* je založené na modelu třídy *Reservation*. To umožňuje lepší typovou kontrolu a jednodušší tvorbu zdrojového kódu. Ve *View* je použito pomocných metod, které poskytuje třída *Html*, pro vytvoření formuláře. Do skrytých polí se ukládají informace potřebné pro vytvoření rezervace. To je uskutečněno skrze pomocnou metodu *HiddenFor* třídy *Html*. Tyto data jsou poté předány při potvrzení formuláře *Controlleru*, který je popsán v kapitole 5.3.

6 Závěr

Pro splnění zadaných cílů této bakalářské práce bylo nutné nastudovat si druhou verzi aplikačního rámce ASP.NET MVC od společnosti Microsoft. Pro pochopení struktury aplikačního rámce ASP.NET MVC 2 je v první části popsán návrhový vzor MVC, který tvoří základ aplikačního rámce ASP.NET MVC 2. Druhá část práce popisuje už samotný aplikační rámec ASP.NET MVC 2, který byl použit pro vývoj konkrétního informačního systému pro správu a rezervaci hudeben. Tato kapitola obsahuje popis nejdůležitějších částí aplikačního rámce ASP.NET MVC 2 a ukázky zdrojových kódu ukazující implementaci jednotlivých částí.

Cílem této práce byla implementace informačního systému pro správu a rezervaci hudeben postavené právě na aplikačním rámci ASP.NET MVC 2. Výsledkem je aplikace obsahující autentizaci a následnou autorizaci uživatele, kterému jsou přidělena různá oprávnění pro využívání služeb hudeben. Dále je zde implementována jednoduchá aplikace rozvrhu, sloužící pro správu rezervací jednotlivých hudeben a zobrazující aktuální informace o stavu hudebny. Systém obsahuje jednoduchou správu novinek, které může vkládat pouze administrátor a které jsou zobrazovány na úvodní stránce aplikace. Díky využití aplikačního rámce je aplikace připravena na pozdější rozšíření nebo na jednoduchou úpravu pravidel pro rezervaci hudeben. Stejně tak může být upravena finální podoba vzhledu webových stránek podle potřeb zadavatele. Aplikace byla navržena tak, aby bylo možné její rozšíření o další hudebny.

Implementace informačního systému probíhala skrze vývojové prostředí Microsoft Visual Studio 2010, které nativně podporuje aplikační rámec ASP.NET MVC 2. Díky této podpoře usnadňuje Visual Studio udržovat správnou strukturu adresářů a tříd, a také pomáhá dodržovat jmenné konvence, které jsou u tohoto aplikačního rámce důležité. Při implementaci informačního systému došlo k výměně aplikačního rámce pro objektově-relační mapování. Místo aplikačního rámce LINQtoSQL bylo využito novějšího aplikačního rámce Entity Framework. Aplikační rámec Entity Framework zjednodušil a zpřehlednil výsledný kód aplikace.

Snahou této bakalářské práce bylo využít vlastností, které aplikační rámec ASP.NET MVC 2 nabízí. V práci byly užity nové prvky, mezi které patří silně typový HtmlHelper, volitelný parametr, nově provedená validace dat na straně serveru pomocí anotací a další.

Jedním z bodů zadání této bakalářské práce bylo srovnání aplikačního rámce ASP.NET MVC 2 s jeho alternativou ASP.NET WebForms. Tato část je zde vypracována velmi okrajově, protože by rozsahem mohla být samostatnou bakalářskou prací.

7 Použitá literatura, zdroje dat a zdrojových kódů

[1] GALLOWAY, Jon, Phil HAACK, Scott HANSELMAN, Scott GUTHRIE a Rob CONERY. *Professional ASP.NET MVC 2*. Indianapolis, IN: Wiley, c2010, 518 s. Wrox professional guides. ISBN 04-706-4318-8.

[2] BOREK, Bernard. Zdroják.cz: MVC a další návrhové vzory. [online]. [cit. 2012-04-25]. Dostupné z: http://www.zdrojak.cz/serialy/mvc-a-dalsi-prezentacni-vzory/

[3] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003, 533 s. ISBN 03-211-2742-0.

[4] *Microsoft ASP.net*: *MVC* [online]. 2012 [cit. 2012-04-25]. Dostupné z: http://www.asp.net/mvc

[5] *Microsoft MSDN* [online]. [2011] [cit. 2012-04-25]. Dostupné z: http://msdn.microsoft.com/cs-cz/library/dd394709.aspx

[6] *Renovace MVC Frameworků* [online]. Brno, 2008 [cit. 2012-04-20]. Dostupné z: http://pisar.wz.cz/Struts1toStruts2/xhtml/dp.xhtml#id2468912. Diplomová. Masarykova universita. Vedoucí práce RNDr. Jan Pavlovič.

[7] FAKULTA INFORMATIKY MASARYKOVY UNIVERZITY. *ASP.NET Tutoriál* [online]. 2010 [cit. 2012-04-25]. Dostupné z: http://kurzy.ucn.muni.cz/DotNet/

[8] ASP.NET. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-27]. Dostupné z: http://cs.wikipedia.org/wiki/ASP.NET

[9] MACDONALD, Matthew, SZPUSZTA, Mario. ASP.NET 2.0 a C# : tvorba dynamických stránek PROFESIONÁLNĚ. Brno, Česká republika : ZONER software, s.r.o., 2006. 1376 s. ISBN 80-86815-38-2

[10] MAREŠ, Vlastimil. *Redakční systém v .NET* [online]. Brno, 2007 [cit. 2012-04-20]. Dostupné z: http://www.fit.vutbr.cz/study/DP/rpfile.php?id=4791 Diplomová. Vysoké učení technické v Brně. Vedoucí práce Ing. Zbyněk Křivka.

[11] PODEŠVA, Tomáš. *Vývoj webových aplikací v Microsoft ASP.NET MVC a WebForms* [online]. Olomouc, 2011 [cit. 2012-04-25]. Dostupné z: http://theses.cz/id/x6p0tb/bcISO.pdf. Bakalářská. Universita Palackého.

[12] *Django*: *Webový framework pro perfekcionisty s termíny* [online]. 2005-2012 [cit. 2012-04-27]. Dostupné z: http://www.djangoproject.cz/

[13] *Root.cz*: *Informace nejen ze světa Linuxu* [online]. 1998 - 2012 [cit. 2012-04-28]. Dostupné z: http://www.root.cz/

[14] *Ruby on Rails*: *Web development that doesn´t hurt* [online]. 2003 - 2012 [cit. 2012-04-30]. Dostupné z: http://rubyonrails.org/

8 Seznam příloh

1. **Případy užití informačního systému pro správu a rezervaci hudeben (5 stran)**

2. **Obsah CD**
   1. Text bakalářské práce (ve formátu pdf)
   2. Spustitelný program
      a. soubor MusicHallMVC.sln spustitelný ve vývojovém prostředí Visual Studio 2010
      b. databáze potřebná pro běh aplikace
      c. zdrojové kódy aplikace
   3. Přístupové informace pro práci s aplikací

Příloha 2: Případy užití informačního systému pro správu a rezervaci hudeben

Registrace uživatele	
Seznam aktérů:	Uživatel, Administrátor
Prekondice:	1. Uživatel musí mít správně vyplněný formulář 2. Uživatel musí souhlasit s pravidly užívání hudeben
Postkondice:	1. Uživatel je zaregistrován v databázi v neaktivním stavu
Typický průběh:	Uživatel vyplní všechna povinná políčka ve formuláři a odešle informace systému pro uložení do databáze
Alternativní průběh:	Pokud jsou nějaká políčka špatně vyplněna, zobrazí se uživateli výzva k opravě. Registrace nebude provedena, dokud je neopraví
Priorita:	Vysoká
Četnost užití:	Hlavně na začátku školního semestru

Tabulka 4 - případ užití registrace uživatele

Potvrzení registrace uživatele	
Seznam aktérů:	Administrátor
Prekondice:	1. Musí být vytvořena registrace uživatele 2. Uživatelský účet nesmí být potvrzen
Postkondice:	1. V databázi se upraví uživatel (povolené hodiny, aktivní stav) 2. V databázi se vytvoří členský poplatek na daný semestr
Typický průběh:	Administrátor přidělí uživateli počet hodin na týden, které slouží pro rezervaci hudebny, a zadá datum přijetí poplatku za využívání služeb.
Alternativní průběh:	Administrátor může aktivovat uživatele i bez zapsání poplatku, ale uživatel nebude schopen rezervovat hudebny.
Priorita:	Vysoká
Četnost užití:	Hlavně na začátku školního semestru

Tabulka 5 – případ užití potvrzení registrace uživatele

Přihlášení uživatele	
Seznam aktérů:	Uživatel Administrátor
Prekondice:	1. V databázi musí existovat uživatelské jméno s daným přihlašovacím jménem a heslem 2. Uživatelský učet musí být potvrzen administrátorem
Postkondice:	1.Uživateli je zpřístupněna aplikace rozvrhu a ostatní části aplikace
Typický průběh:	Uživatel vyplní uživatelské jméno a heslo. Po ověření je přihlášen do systému.
Alternativní průběh:	Při špatném zadání uživatelského jména a hesla systém zamítne uživateli přístup do systému a vyzve ho k opakování přihlášení.
Priorita:	Vysoká
Četnost užití:	několikrát za den

Tabulka 6 - případ užití přihlášení uživatele

Změna stavu hudebny	
Seznam aktérů:	Administrátor
Prekondice:	1. Hudebna musí existovat
Postkondice:	1. V databázi se změní stav hudebny a přidá důvod změny stavu
Typický průběh:	Administrátor změní stav hudebny a zapíše důvod změny
Alternativní průběh:	Administrátor změní stav hudebny a nezapíše důvod
Priorita:	nízká
Četnost užití:	jednou do měsíce

Tabulka 7 - případ užití změna stavu hudebny

Udělení ban uživateli	
Seznam aktérů:	Administrátor
Prekondice:	1. Uživatel musí existovat v databázi
Postkondice:	1. Uživateli je zapsán záznam o udělení banu do databáze 2. Všechny rezervace v době trvání zamezení přístupu jsou zrušeny a zpřístupněny ostatním uživatelům
Typický průběh:	Administrátor zvolí uživatele, kterému omezit přístup do hudeben. Pří udělení banu administrátor zapíše datum, do kdy je platný, a důvod proč byl uživateli udělen.
Alternativní průběh:	Administrátor bude upozorněn při chybějících údajích.
Priorita:	střední
Četnost užití:	měsíčně

Tabulka 8 - případ užití udělení ban

Přidání požadavku	
Seznam aktérů:	Uživatel
Prekondice:	1. Uživatel musí být přihlášen
Postkondice:	1. Požadavku je automaticky přidán status NEW a aktuální datum vložení
Typický průběh:	Uživatel zadá požadavek na lepší vybavení hudebny a podobně. Ten uložen do databáze a je k dispozici na stránce se seznamem požadavků, kde je možné sledovat jejich stav
Alternativní průběh:	Při vložení nepovolených znaků do popisu se požadavek neuloží.
Priorita:	nízká
Četnost užití:	měsíčně

Tabulka 9 - případ užití přidání požadavku

Změna stavu požadavku	
Seznam aktérů:	Administrátor
Prekondice:	1. Požadavek musí existovat
Postkondice:	1. U požadavku je změněn status a poté je uložen zpět do databáze
Typický průběh:	Administrátor změní status vybranému požadavku.
Alternativní průběh:	
Priorita:	nízká
Četnost užití:	měsíčně

Tabulka 10 - případ užití změna stavu požadavku

Zobrazení stavu hudebny	
Seznam aktérů:	Uživatel, Administrátor
Prekondice:	1. Uživatel musí být přihlášen do systému
Postkondice:	
Typický průběh:	Uživateli se zobrazí aktuální stav hudebny a její aktuální obsazení.
Alternativní průběh:	Pokud není aktuálně obsazena, zobrazí se pouze stav hudebny.
Priorita:	střední
Četnost užití:	několikrát za den

Tabulka 11 - případ užití zobrazení stavu hudebny

Zobrazení novinek	
Seznam aktérů:	Uživatel, Administrátor, nepřihlášený uživatel
Prekondice:	1. V systému musí být uloženy novinky
Postkondice:	
Typický průběh:	Při přístupu na web je zobrazena uživateli úvodní stránka s výpisem všech novinek.
Alternativní průběh:	Pokud nejsou v systému vedeny žádné novinky, zobrazí se uživateli prázdná stránka.
Priorita:	vysoká
Četnost užití:	několikrát za den

Tabulka 12 - případ užití zobrazení novinek

Přidání novinky	
Seznam aktérů:	Administrátor
Prekondice:	
Postkondice:	Novinka je automaticky zobrazena na první pozici úvodní stránky webu
Typický průběh:	Administrátor do formuláře zadá titulek novinky a samostatnou zprávu a po odeslání je zpráva uložena do databáze.
Alternativní průběh:	Pokud administrátor přesáhne počet znaků, informační systém ho na to upozorní a neuloží novinku do databáze.
Priorita:	vysoká
Četnost užití:	týdně

Tabulka 13 - případ užití přidání novinky

Upravení novinky	
Seznam aktérů:	Administrátor
Prekondice:	1. Daná novinka musí být evidována v systému
Postkondice:	1. U novinky zůstává původní datum zadání do systému.
Typický průběh:	Administrátor může zprávu nebo titulek novinky.
Alternativní průběh:	Při přesáhnutí maximálního počtu znaků zprávy nebo titulku je administrátor upozorněn na tuto skutečnost a změna není uložena do databáze
Priorita:	vysoká
Četnost užití:	měsíčně

Tabulka 14 - případ užití upravení novinky

Smazání novinky	
Seznam aktérů:	Administrátor
Prekondice:	1. Novinka musí být evidována v informačním systému 2. Novinka již nemá platnost.
Postkondice:	1. Novinka je smazána z databáze
Typický průběh:	Administrátor smaže vybranou novinku, tím se zamezí její zobrazení na titulní stránce.
Alternativní průběh:	
Priorita:	vysoká
Četnost užití:	měsíčně

Tabulka 15 - případ užití smazání novinky